Modeling and verification of probabilistic data-aware business processes

Haizhou Li

N° d'ordre : D. U : 2563
E D S P I C : 694

UNIVERSITE BLAISE PASCAL - CLERMONT II

ECOLE DOCTORALE
SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND

Modélisation et Vérification des Processus Métier Orientés Données Probabilistes

Présentée par

Haizhou LI :

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

SPECIALITE : Informatique

Titre de la thèse :
Modélisation et Vérification des Processus Métier Orientés Données Probabilistes

Soutenue publiquement le 26/03/2015 devant le jury :

Mme	Salima Benbernou	Rapporteur et examinateur
Mme	Karine Zeitouni	Rapporteur et examinateur
M .	Vasile Marian Scuturici	Examinateur
M .	Sandro Bimonté	Examinateur
M .	Farouk Toumani	Directeur de thèse
M .	François Pinet	Directeur de thèse

Université Blaise Pascal

Doctoral Thesis

Modeling and Verification of Probabilistic Data-aware Business Processes

Author:
LI Haizhou

Supervisor:
TOUMANI Farouk
PINET François

[1] *A thesis submitted in fulfilment of the requirements for the degree of Computer Science*

in the

LIMOS, CNRS

April 10, 2015

[1]The author is funded by "La région Auvergne et le Feder".

UNIVERSITÉ BLAISE PASCAL

Abstract

LIMOS, CNRS

Doctor of Philosophy

Modeling and Verification of Probabilistic Data-aware Business Processes

by LI Haizhou

There is a wide range of new applications that stress the need for business process models that are able to handle imprecise data. This thesis studies the underlying modelling and analysis issues. It uses as formal model to describe process behaviours a labelled transitions system in which transitions are guarded by conditions defined over a probabilistic database. To tackle verification problems, we decompose this model to a set of traditional automata associated with probabilities named as world-partition automata. Next, this thesis presents an approach for testing probabilistic simulation preorder in this context. A complexity analysis reveals that the problem is in 2-EXPTIME, and is EXPTIME-hard, w.r.t. expression complexity while it matches probabilistic query evaluation w.r.t. data-complexity. Then P-LTL and P-CTL model checking methods are studied to verify this model. In this context, the complexity of P-LTL and P-CTL model checking is in EXPTIME. Finally a prototype called "PRODUS" which is a modeling and verification tool is introduced and we model a realistic scenario in the domain of GIS (graphical information system) by using our approach.

Key words: probabilistic database, business processes, simulation relation test, model checking.

Résumé

Un large éventail de nouvelles applications met l'accent sur la nécessité de disposer de modèles de processus métier capables de manipuler des données imprécises ou incertaines. Du fait de la présence de données probabilistes, les comportements externes de tels processus métier sont non markoviens. Peu de travaux dans la littérature se sont intéressés à la vérification de tels systèmes. Ce travail de thèse étudie les questions de modélisation et d'analyse de ce type de processus métier. Il utilise comme modèle formel pour décrire les comportements des processus métier un système de transitions étiquetées dans lequel les transitions sont gardées par des conditions définies sur une base de données probabiliste. Il propose ensuite une approche de décomposition de ces processus qui permet de tester la relation de simulation entre processus dans ce contexte. Une analyse de complexité révèle que le problème de test de simulation est dans 2-EXPTIME, et qu'il est EXPTIME-difficile en termes de complexité d'expression, alors que du point de vue de la complexité en termes des données, il n'engendre pas de surcoût supplémentaire par rapport au coût de l'évaluation de requêtes booléennes sur des bases de données probabilistes. L'approche proposée est ensuite étendue pour permettre la vérification de propriétés exprimées dans les logiques P-LTL et P-CTL. Finalement, un prototype, nommé 'PRODUS', a été implémenté et utilisé dans le cadre d'une application liées aux systèmes d'information géographiques pour montrer la faisabilité de l'approche proposé.

Mots-clés: bases de données probabilistes, processus métier, relation de simulation, vérification de modèles.

Acknowledgements

I am forever indebted to Farouk TOUMANI. Farouk is blessed with an astonishing combination of brilliance and patience, and I will be forever grateful for the skills he has patiently taught me. I owe him my research career, which is one of the most rewarding pursuits in my life. François PINET offered me patient support and advice. I am profoundly grateful for his taught in the domain of GIS which is an unfamiliar field for me and brilliant advices. I appreciate his patience and erudition. The students in the database group and the Axe 2 Systèmes d'Information et de Communication made my graduate studies both intellectually stimulating and incredibly enjoyable. An incomplete list of those who helped me on this path include John SAMUEL, Hicham REGUIEG, Lakhdar AKROUN, Karima ENNAOUI. The conversations that I had with each one of them enriched my life and made this work a joy to pursue. The faculty and staff at the University of Blaise Pascal have created a wonderful place, and I am very thankful to be a part of it. Special thanks to Libo REN and Xinran LI. They gave me so many excellent advices about living in Clermont-Ferrand and made me not feel alone in Chinese festivals. Finally, I would like to thank my family. My parents, not only provided love and support thoughout my life, but also listened to hours of chatter about my problems living in France. I am, however, most grateful to my girlfriend Siqi HAN. Siqi's patience, love, and support are boundless. For any current or future success that I might have, Siqi deserves more credit than I can possibly express in a few lines of text.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the domain of Business Process Management (BPM)[1, 2], the core is process modelling and analysis. A business process is usually defined as a collection of activities performed in coordination to achieve a particular business goal [1]. The uses of business process models have been spread from business scope (e.g., supply chain system) to many other domains (e.g., agriculture, medicine). Following the numerous nouveau demands, various process models [1, 2] have been proposed in the literature to describe business processes ranging from theoretical models, which implement formal semantics (e.g., transition systems, Petri nets, process algebras, EPC), to executable models (e.g., BPEL, XPDL, WS-CDL) or graphical ones (e.g., BPMN, activity diagrams of UML). While most of existing models focus on control-flow perspective (i.e., ordering and coordination of activities), which is an essential dimension to describe business processes, there has been over the last few years an increasing interest around the role played by data in business processes. Indeed, in many applications the executions of processes, as specified in a control-flow, may be also governed by conditions defined over variables or over a database. This motivates the emergence of data-aware and data-centric perspectives for process modelling, approaches that promote data to first-class citizens in process models. The interest in these perspectives is driven by many applications in which data plays a prominent role such as artifact-centric modelling of business processes [3, 4], data-aware conformance and compliance checking [5, 6] of business processes as well as data-centric web services [7].

However, whereas many traditional applications manipulate precise data, there is a wide range of new applications that need to manage imprecise and uncertain data [8]. Many sources of imprecision of data are possible, among them the following examples are worth mentioning:

- Nowadays, most business processes are collaborative, span across enterprises boundaries and, as a consequence, have to deal with data originated from multiple sources. This raises information quality issues since data sources are usually heterogeneous and methods and frequency of collecting data vary depending on organizations and geographies [9]. In the sense that a same real world entity may be represented differently in different sources. Furthermore, data may be *"collected with different methods and frequency by different departments, institutions, and geographies"* [9]. Data cleaning and source reconciliation are costly tasks, in particular if the size of data is very large. As a consequence, in most of the state-of-the-art, data integration approaches allow the data to be imprecise. For example, as highlighted in [8], in business intelligence, imprecision is the price to pay to reduce the cost of data cleaning. While the author of [9] explains that *many possible sources of uncertainty exist in BI applications, and this problem is magnified when data comes from multiple sources and is collected with different methods and frequency by different departments, institutions, and geographies.*

- Recently, several research works [10, 11] as well as industrial tools (e.g., Oracle's BPEL Sensor and IBM WebSphere Business sensor events) highlighted the need of integrating sensor network data into business processes. Example of applications include fleet management and package tracking related to logistics processes [10] or monitoring processes where sensors and actuators driven by a business process are used to control large-scale physical systems (e.g., energy efficient buildings [11]). Data captured from the physical world using sensors, cameras is inherently imprecise and uncertain.

- In many applications areas, e.g., healthcare, financial services or business intelligence, data is very sensitive and cannot be handled as it is by business processes. Privacy regulations may impose various requirements such as anonymity, data masking, obfuscation, or introduction of imprecision, (e.g., noise), to hide sensitive information [8].

The aforementioned applications stress the need for business process models that are able to handle imprecise data. We study in this thesis the underlying modelling and analysis issues. We use as formal model to describe process semantics a Labelled Transitions System (LTS) in which transitions are guarded by conditions defined over a global database which, in spirit of [8], *contains an explicit representation of the uncertainty.* We call such a model a *probabilistic data-aware business process* (pd-process). Our choice of LTSs is motivated by the prominent role played by this formalism for representing behaviours of systems. Indeed, LTSs form one of the most used types of models in process theory [12] and they have also been used intensively in BPM and web services areas to

support formal analysis of business processes [13]. We rest on recent developments in the emerging field of probabilistic databases [8, 14, 15] to include imprecise data within labelled transitions systems and formally define the semantics of the obtained pd-process model.

This dissertation focuses on the problem of modelling and analysing pd-processes. Historically, two principal methods of equal importance have been used in the literature to analyse LTSs: temporal logic, used to verify whether a given process satisfies certain properties and equivalence or preorder relations. In the first class, Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) are normally considered to verify process models. LTL is a logical formalism capturing linear time properties and it is a fragment of CTL which allows branching time and qualifiers. By using these temporal logic, model verification or called model checking is wildly used to help process designers to detect the defects of a model effectively by verifying essential properties. Regarding the second class of methods, simulation preorder is a refinement relation on processes that has been proved to be very useful in many applications. Simulation equivalence plays a crucial role in model checking since it preserves relevant properties of many temporal logics (e.g., CTL*) and hence can be exploited to minimize the state space explored by verification algorithms [16, 17]. Simulation equivalence has also been used directly for verification of business processes [18] as well as for web service analysis and composition [19, 20].

1.1 Contribution

In this thesis, our contributions are as follows:

- We integrate probabilistic database with automata theories to propose a nouveau process model, named "Probabilistic data-aware business process model", which expands the usage domain of business process model. This model specifies boolean queries on probabilistic databases as guards which enrich the decision making and bring another way to express probabilities in a labelled transition system. Indeed, the general topic of this thesis is not totally new since a satisfactory verification theory for probabilistic processes has been a long-standing research problem and numerous probabilistic process models have already been proposed in the literature [21]. Whereas most existing models assume a form of independence between transition probabilities, in pd-processes there is an intricate correlation between transitions due to the presence of guards over a probabilistic database. As a consequence, pd-process semantics does not coincide with semantics of probabilistic

processes previously described in the literature, which makes it difficult to reuse existing techniques to do simulation relation test in pd-processes.

- We propose a refined approach to decompose a pd-process into a set of world partition automata which can be seen as traditional automata with probability distribution. This mechanism helps us to tackle verification problems of pd-processes.

- A formal definition of simulation relation preorder in the context of pd-processes is made into two dimensions: (i) semantic, in the sense that it is based on a containment relation between the possible execution trees of pd-processes, and (ii) conservative, since it matches classical notion of simulation in non-probabilistic case and a refinement approach that enables to characterize simulation preorder in pd-processes. With the help of world partition automata, we can reuse traditional method of simulation relation and model checking to verify the model of pd-processes. The complexity of simulation test is studied in the dimensions of expression complexity and data complexity. We prove that the size of probabilistic database does not produce any overhead w.r.t the data complexity and establish upper and lower bound of expression complexity.

- By reusing the traditional model checking algorithms in the literature, we produce a sort of verification algorithms in the context of linear temporal logic (LTL) and computational tree logic (CLT) as well as their probabilistic counterpart Probabilistic-LTL (P-LTL) and Probabilistic-CTL (P-CTL). Owing to the refinement of pd-processes a pd-process can be reconstructed with a set of normal automata where every automaton is associated with a probability. So the traditional model checking algorithms in the context of LTL or CTL can be applied in the refinement structures of pd-processes. Then the complexity of LTL model checking on pd-processes is proved that there is no overhead w.r.t the complexity of traditional LTL model checking but the CTL model checking on pd-processes reaches exponential.

- We provide several optimized simulation relation algorithms of pd-processes. Firstly, we classify pd-processes to three scopes: deterministic pd-processes, pseudo deterministic pd-processes and strong non-deterministic pd-processes. If a pd-process is a deterministic pd-process or a pseudo-deterministic pd-process, the complexity of testing simulation relation is in EXPTIME. Besides, we introduces a special case: the simulation relation test on a compiled approach when only the content of query or database is altered.

- A pd-process can be considered as a Markov process if it satisfies some properties. We propose the notion of independent pd-processes to build the bridge of linking

pd-processes and Markov processes. Meanwhile, the complexity of simulation test and model checking over independent pd-processes can be decreased, for example: the complexity of simulation relation on independent pd-processes is in PTIME rather than EXPTIME.

- We design and implement a probabilistic data-aware business process framework abbreviated as **PRODUS** which is a pd-process model and verification tool. **PRODUS** provides a graphical user interface to visualize designers' process building and links a probabilistic system which is powered by PostgreSQL. Meanwhile, **PRODUS** is capable to test the preorder relation and model verification (LTL, CTL, P-LTL and P-CTL) in the context of pd-processes. Due to the lack of the intersection algorithm of probabilistic boolean queries in the literature, an algorithm of intersecting probabilistic boolean queries is devised to fulfil the needs from preorder relation test.

1.2 Overview of Dissertation

This dissertation is organized as follows:

- Chapter 2 introduces some background knowledge: (i) probabilistic database, this part briefly depicts the semantic of probabilistic database in the sense of possible worlds and an example related to the car insurance risk analysis introduced in the motivation, (ii) simulation relation testing algorithm, the traditional preorder relation testing algorithm by [22], (iii) model checking algorithms, in this subsection, the model verification algorithms in the context of LTL, P-LTL, CTL, P-CTL are introduced respectively, including their syntax, semantics, algorithms and complexity discussions.

- In Chapter 3, we propose a formal definition of probabilistic data-aware business process, and we discuss its semantic in terms of possible execution trees. The main concepts of pd-processes are explained through an example of car insurance risk analysis. Then we refine the structure of pd-processes to a finite set of partition automata which will be useful to facilitate verification algorithms for pd-processes.

- The probabilistic simulation relation algorithm is described in Chapter 4. This chapter provides the definition of simulation relation preorder in the context of pd-process in terms of possible execution trees but this definition cannot devise a direct algorithm because the number of possible execution trees may be infinite. Then, we discuss the complexity of this preorder relation algorithm in two dimensions:

expression complexity in terms of the size of pd-processes and data complexity w.r.t the volume of probabilistic database. Meanwhile, Chapter 4 describes the model checking methods of pd-processes in the context of various temporal logics: LTL, P-LTL, CTL, P-CTL. By reusing the refinement structure of pd-processes the traditional model verification algorithms can be used with slight modifications.

- Chapter 5 discusses several optimized algorithms of simulation relation test over pd-processes. Finally, we study the relation between pd-processes and Markov processes. The notion of independent pd-processes reveals a fact that if a pd-process satisfies some properties, it is a Markov process: complexity of verification can be reduced.

- Chapter 6 discusses the related work and the differences between these works with our contributions.

- Chapter 7 introduces the prototype **PRODUS**, the modelling of agricultural environmental risk analysis and several experiment result: system performance of simulation relation test by inputting synthetic data and realistic information from the agricultural scenario, model checking results of this scenario by inputting some properties in terms of LTL, P-LTL, CTL and P-CTL respectively.

Chapter 2

Preliminary

This chapter provides several pieces of background material for this dissertation: the notion of probabilistic database, a brief introduction about the labelled transition system which is specified as finite state machine in this thesis, the notion of Markov processes, the definition of simulation preorder relation, and the theories of model checking.

Motivation example

In this thesis, we consider an example of a business process used by an insurance company as a running example. A main business goal of a company is to ensure that revenues must be greater than expenses. In the case of an insurance company, a large part of revenues come from customer premiums while the largest expenses are related to the payment of claims. Therefore, calculation of insurance rates is of high importance and depends on a number of different factors. The main business objective is to calculate premiums that are adequate, in order to maintain company solvency, and which are as much as possible fair compared to the risk, in order for example to avoid loosing clients who may be attempted to move to cheaper competitor companies. Now the problem is how to set a reasonable price for every candidate. Normally, car insurance companies will let their insurance applicants to fill a complex form with a variety of personal information, such as driving history, gender, age, education level, and record this information in the database. Then the statistic should find the association between these driver personal information with the financial risk. If we assume that all this information correctly reflect the truth situation of candidates, we could approximate the probabilities of a combination of principle criteria impacting the finance risk. Then the problem comes to us, how can we build an effective business process and reflect the probabilistic data influencing the decisions in this business process.

Teacher	Course
Tom	History
@	French
Pascal	@

Teacher	Course
Tom	History
⟨Mary,Susan⟩	French
Pascal	⟨ Biology, Chemistry⟩

TABLE 2.1: Example of incomplete databases

2.1 Probabilistic database

The issues underlying management of imprecision and uncertainty in data have attracted the attention of the research community since a long time. Several models have been proposed over the time to handle uncertain data [9, 23]. In recent years, the field of probabilistic databases gained momentum under the driving force of a wide spectrum of new applications [8, 14, 15].

Because relational database is designed to be capable to store incomplete data at the very beginning, the model and theory of incomplete database was birthed to fulfil the needs of representing and querying incomplete data which is stored in relational databases. The original work was captured in [24] on Codd, c-, v-tables with their conditional tables and introduced the notion of representation system. Intuitively, incomplete database is a relational database which contains incomplete data (missing data as well as "Or-set" value). Table 2.1 illustrates an example of different presentation of incomplete databases: Null value and "Or-set" value. In the left table, @ represents null value and this table is a Codd table [24]; ⟨Mary,Susan⟩ is a "Or-set" value, showing that both "Mary" or "Susan" are possible values. Incomplete database can be seen as a (infinite) set of complete databases. These complete databases represent states of real world. The incomplete database may be infinite and hard to manipulate. So we need a tool to describe the infinite databases in a finite way which we could manipulate and query. Because of this reason, a representing sytem is created to represent the incomplete information. We denote the $< T, rep, \Omega >$ as a representing system. T stands for a set of multi tables $< T_1, T_2, \ldots T_n >$ and it can be used like a model to represent the incomplete information. $rep(T)$ is denoted as the set of possible relations that T represents. rep defines a mapping from T to $rep(T)$ or we can say a mapping from $Tab(R)$ to $Inc(R)$. If R is a relation scheme, $Tab(R)$ presents the set of all tables on R and $Inc(R)$ is the power set of all the set of relations on R. Ω stands for a set of relational operators such as projection, selection, etc. After the work of Imielinski and Lipski [24], later work on incomplete database has focused on the notion of possible or certain answers [25], completeness of query [26]. Meanwhile, the study of incomplete information is not only limited in the domain of relational database but also extended to XML file system [27].

By extending the notions of incomplete database, probabilistic database is birthed to manage probabilistic data. The original work of probabilistic database can be derived in 80s in [28] where attributes are random variables. Then in [29], the probability was considered as an uncertain value of an attribute and the method of evaluation of select-join queries was also studied in this paper. Later on, the semantic of possible worlds were defined in [30] and an approach to ensure efficient query evaluation by relaxing the probabilistic semantic was studied in [31]. Then the concept of probabilistic schema (by-table or by-tuple) was described in [32]. Nowadays, there has been plenty of researches over probabilistic database by extending previous result to expand the field of probabilistic database [8, 14, 15].

The application of probabilistic database currently focused on managing uncertain scientific information. In [33], a Probabilistic Tree Database based on a probabilistic XML model has been designed to fulfil the needs in the domain of biology. Later on, an application named BioRank was proposed in [34] for exploratory queries on tracking the uncertainties by joining information from various data source . This system is used to predict protein functions and considers the uncertainty in the domain of scientific data integration. They revealed the advantage of consideration of probabilities to handle vogue problem and to manage uncertain data. Currently, there are various project relating to probabilistic database such as Trio [23] which manages uncertain data and data lineage, MystiQ [8] that is a probabilistic query evaluation prototype. Then flurry of DBMS of probabilistic database are developed, for example the systems for conjunctive queries: MayBMS [35], PrDB [36], ORION [37], and SPROUT [38] which supports full relational algebra. Next section will present another application "Spatial relation" which uses probabilistic database to manage uncertainty over agricultural plots with vague boundaries in the domain of geography.

The remaining section introduces some basic concepts of the theory of probabilistic databases [8, 14, 15]. We assume the reader familiar with basic database concepts (e.g., see [39] for details). In this thesis, we are interested in particular by probabilistic relational databases defined over a finite domain. Informally, a probabilistic database is defined as a database that includes relations whose tuples are associated with probabilities.

Definition 2.1. A finite probability space is a pair (Ω, Pr) where Ω is the finite set of *outcomes*, and $Pr : \Omega \to [0,1]$ s.t. $\sum_{\omega \in \Omega} Pr(\omega) = 1$. For $A \subseteq \Omega$, we take $Pr(A) = \sum_{\omega \in A} Pr(\omega)$. A set $\{t_1, ..., t_n\} \subseteq \Omega$ is independent if $Pr(t_1, ..., t_n) = Pr(t_1) \times \ldots \times Pr(t_n)$.

In the example of car insurance company, to evaluate the risk and to help managers to insight the homogeneous criteria of candidates, the probabilistic database is an ideal tool

Relation Applicant

	ID	Name	Age	Educ. Lev	Lic Year	Driv.Rec	City	Pr
t_1	C101	Jack	20	college	3	medium	paris	1

Relation Profit

	AgeMin	AgeMax	Educ. Lev	Lic Year	Driv. Rec	City	Profit	Pr
t_2	25	50	college	3	medium	Paris	high	60%
t_3	20	30	college	3	medium	Paris	medium	50%
t_4	18	25	college	3	none	Paris	low	60%
t_5	45	70	high school	10	minor	Lyon	high	70%

Relation Risk

	AgeMin	AgeMax	Educ. Lev	Lic Year	Driv. Rec	City	Lev.Risk	Pr
t_6	18	25	college	3	medium	Paris	high	30%
t_7	20	35	college	3	medium	Paris	medium	80%
t_8	30	55	college	3	medium	Paris	low	40%
t_9	45	70	high school	10	minor	Lyon	high	50%

TABLE 2.2: Example of a probabilistic database (D_{ins}).

to reveal the relation of different attributes combinations. Table 2.2 shows an example of a probabilistic database[1], noted D_{ins}, in the field of insurance risk assessment. The database D_{ins} contains three relations: Applicant, Profit and Risk. The Applicant relation is used to store information about an application of a new customer. The Profit relation is used to record the profit insurance companies forecast depending on drivers profiles such as age, level of education, number of licence years and city. The relation Risk records statistical information about levels of financial risk in association with drivers profiles. Such information could be, for example, computed from an analysis of the history of claims maintained by an insurance company. The content of the relation Profit and Risk is indeed not certain and, hence, the two relations record a confidence with each prediction or analysis result. This is materialised by the attribute Pr in each relation which associate a probability with each tuple in a (probabilistic) relation (i.e., Pr gives the marginal probability of each tuple). In this example, profit forecast reveals that senior drivers living in the city of Lyon are likely to generate a high level of profit. This information is captured by the tuple t_5 of the relation Profit which has a probability equal to 70%. The standard semantics of probabilistic databases is defined based on the notion of *possible worlds*. The intuition is that the precise content of a probabilistic database is unknown but instead the finite set of potential instances, each with some probability, can be computed. Continuing with the previous example, Table 2.3 shows some possible worlds (i.e., instances) of the probabilistic database D_{ins} (the total number of all the possible worlds is 2^8). Hence, a probabilistic database can be viewed as a probabilistic distribution over a finite set of possible (complete) databases. Given a probabilistic database D, we denote by $\mathcal{W}(D)$ the finite set of

[1]In this example, and only for illustration purposes, tuples are assumed to be independent (e.g., this is why the sum of probabilities of the tuples t_2 and t_3 of relation Profit is > 1). Such an assumption is not mandatory for the proposed approach.

World ID	Possible World	Pr
W_1	ϕ	0.1008%
W_2	$Applicant = \{t_1\}, Profit = \{t_2, t_4\}, Risk = \{t_7\}$	0.9072%
W_3	$Applicant = \{t_1\}, Profit = \{t_2, t_4\}, Risk = \{t_6, t_7\}$	0.3888%
W_4	$Applicant = \{t_1\}, Profit = \{t_2, t_3, t_4\}, Risk = \{t_7\}$	0.9072%
W_5	$Applicant = \{t_1\}, Profit = \{t_3\}, Risk = \{t_6\}$	0.0648%
W_6	$Applicant = \{t_1\}, Profit = \{t_5\}, Risk = \{t_6\}$	0.1008%
W_{256}	$Applicant = \{t_1\}, Profit = \{t_2, t_3, t_4, t_5\}, Risk = \{t_6, t_7, t_8, t_9\}$	0.6048%

TABLE 2.3: Some possible worlds of the probabilistic database D_{ins}.

its possible worlds (i.e., its possible instances). Formally, a probabilistic database D defines a finite probability space $(\mathcal{W}(D), Pr)$, whose set of outcomes $\mathcal{W}(D)$ form all the possible instances of the probabilistic database D. Each possible world $W \in \mathcal{W}(D)$ is associated with a probability given by $Pr(W)$, with $\sum_{W \in \mathcal{W}(D)} Pr(W) = 1$.We recall that conjunctive queries correspond to queries that can be formulated using the Select-Project-Join operators of the relational algebra and they form the most used fragment of existing relational query languages. We focus our attention on boolean queries,

Given such a framework, a crucial question is then related to query evaluation, i.e., *the problem of calculating the probability of tuples occurring in query answers* [15]. In this thesis, we are interested in particular by *boolean queries*, i.e., queries that return as unique answers either true or false. When needed in the examples, we use a Datalog-like notation to write boolean queries [39]. For example, using the database depicted at Table 2.2, the query:

$$\begin{aligned} q_1() \quad :\text{-} \quad & Applicant(\_, \_, A, \_, \_, \_, \_), Risk(A1, A2, \_, \_, \_, \_, R), \\ & A1 \leq A \leq A2, R =' high' \end{aligned}$$

expresses a join between the relation Applicant and the tuples of the relation Risk having a level of risk equal to 'high' and using as join condition the ages of applicants which must be included between the min and max ages of the relation Risk. In the query q_1, the letters A, $A1$, $A2$ and R denote variables while the symbol $'\_'$ is used to denote *anonymous variables*. When evaluated on a conventional database instance I, the query q_1 returns true if there is at least one tuple of Applicant that can be joined with an adequate tuple of Risk (in this case we write $\mathsf{q}_1(\mathsf{I}) = \mathsf{true}$). Otherwise, the query q_1 returns false (i.e., $\mathsf{q}_1(\mathsf{I}) = \mathsf{false}$). In the context of a probabilistic database $(\mathcal{W}(D), Pr)$, the problem of the evaluation of a boolean query q consists in computing the probability of query q to return as answer the value true. Such a probability is defined as follows $Pr(q) = \sum_{\substack{W \in \mathcal{W}(D) | \\ q_1(W) = true}} Pr(W)$. In other words, $Pr(q)$ is given by the sum of the probabilities of all the possible worlds where q is evaluated to true. However, in most practical situations it is not feasible to compute the set $\mathcal{W}(D)$ and then explicitly evaluate a query q on

each world in $\mathcal{W}(D)$. Indeed, $\mathcal{W}(D)$ is usually very large (e.g., in our toy probabilistic database of Table 2.2, the number of possible worlds is already 2^8). To cope with this problem, existing works have developed techniques to efficiently evaluate queries on concise representations of probabilistic databases [8, 14, 15]. Not surprisingly, there is a trade-off between the expressiveness of the representation model and computational tractability of query evaluation. This is why, most existing works adopt some restricting assumptions, often expressed as a form of independence of tuples [8]. There are also some few approaches that support modelling complex correlations in probabilistic databases [14, 15].

It is worth mentioning that, while we rely on existing techniques to handle probabilistic data, our approach remains insensitive w.r.t. the assumptions underlying the representation model. We require only a system that is able to evaluate boolean queries over a probabilistic database, a requirement which is within the reach of most existing probabilistic database management systems.

2.2 Finite state machine

The definition of a finite state machine is given as follows. A finite state machine is a tuple $A = (S, s_0, Act, \Delta, F, L, AP)$, where:

- S is a finite set of states, with $s_0 \in S$, the starting state.
- Act is a finite set of actions or activities.
- $\Delta \subseteq S \times Act \times S$, the transition relations, is a set of guarded transitions.
- $F \subseteq S$ is the set of final states.
- AP is a set of atomic propositions.
- L is a function $L : S \to 2^{AP}$.

$s_0 \xrightarrow{a} s_1$ stands for the transition relation from s_0 to s_1 assigned by action a. The intuition of an automaton can be depicted as follows. Initial state s_0 is the start of the system and F is a set of final states which are the terminator of the system. Δ is the set of transition relations which represent a fact that: if s is the current state, then a transition $s \xrightarrow{a} s'$ originating from s is selected non-deterministically and taken, then the action a is performed and the transition system evolves from state s into the state s'. This selection procedure will be repeated until a final state is encountered. Meanwhile, this procedure is non-deterministic when a state has more than one outgoing transition.

Function L relates a set $L(s) \in 2^{AP}$ of atomic propositions to any state $s_i \in S$. $L(s)$ represents the satisfaction of those atomic propositions $ap \in AP$ by corresponding state s. Noticing that one state may be expressed by one or many atomic propositions. For a state s_i $L(s_i) = \{ap_{i_1}, ..., ap_{i_j}\}$. A path π is a finite sequence of states, actions and transition relations $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 ... \xrightarrow{a_{n-1}} s_n$. We can also use $\pi[0, ..., n]$ represent the whole path and $\pi[i, ..., j]$ stand for a sub path $s_i \xrightarrow{a_i} s_{i+1} ... \xrightarrow{a_{j-1}} s_j$. $Paths(s)$ represents a set of outgoing paths from state s.

2.3 Markov processes

Following the description in [40], the notions of discrete probability space and Markov processes are defined as follows.

Definition 2.2 (Probability space)**.** A probability space is a triple (Ω, F, Pr) where Ω is a set, F is a collection of subsets of Ω, and Pr is a function from F to $[0, 1]$ such that $Pr(\Omega) = 1$ and for any collection $\{C_i\}_i$ of at most countably many pairwise disjoint elements of F, $Pr(\bigcup C_i) = \sum_i C_i$.

A probability space Ω, F, Pr is discrete if $F = 2^{\Omega}$ and for each $C \subseteq \Omega$, $Pr(C) = \sum_{x \in C} Pr(x)$, x is an element of C.

Definition 2.3 (Markov process)**.** A Markov process A is an automaton $(S, s_0, Act, \Delta,$ $Probs(S \times Act))$ whose transition relation Δ is a subset of $S \times Probs(S \times Act)$ where $Probs(S \times Act)$ is a discrete probability space (Ω, F, Pr) such that $\Omega \subseteq S \times Act$.

From the definition of Markov processes, the transitions from the same state are exclusive and the ones from same state are independent, meaning that the probability of current state never influence the predictions of successor transitions.

2.4 Simulation preorder relation

A simulation preorder relation test is to check the containment between two finite state machines. According to [22], the definition of simulation preorder relation is defined as follows. Let $A = (S, s_0, Act, \Delta, F, L, AP)$ and $A' = (S', s'_0, Act', \Delta', F', L', AP')$ be two finite state machines. Then, A is simulated by A', noted $A \lesssim A'$, iff: $\forall s_i \in S$, $s_i \xrightarrow{a_i} s_j$, $\exists s'_i \in S'$ such that $s'_i \xrightarrow{a'_i} s'_j$, $a_i = a'_i$. The algorithm of simulation relation test[22] is represented in Appendix.

2.5 Model checking

Model checking [17] is a technique which automatically checks a given system to satisfy some properties by converting these properties from natural language to temporal logics. The underlying nature of time in temporal logics can be either linear or branching. In the linear view, at each moment in time there is a single successor moment, whereas in the branching view it has a branching, tree-like structure, where time may split into alternative courses. Both temporal logics provide us a different perspective to consider the properties of a system. In this thesis, the methods of model checking will focus on linear temporal logic (LTL), probabilistic linear temporal logic (P-LTL), computation tree logic (CTL) and probabilistic computation tree logic (P-CTL). Following the methods described in [17], the traditional algorithms of model checking as well as syntax and some important notions are depicted in this section. To be clear, the operators of model checking are firstly explained as follows.

- $\bigcirc$:next;
- U:until;
- $\Box$:always (now and for ever in the future);
- $\Diamond$:eventually (eventually in the future).

2.5.1 LTL model checking

Linear temporal logic (LTL) is a logical formalism specifying linear temporal properties to check the satisfaction of paths in a finite state system. The syntax of LTL is depicted as follows:

$$\Phi \vDash true|ap|\varphi_1 \wedge \varphi_2|\neg\varphi|\bigcirc\varphi|\varphi_1 \cup \varphi_2 \tag{2.1}$$

The semantic of LTL formulae is depicted as follows:

- $\pi \vDash true$.
- $\pi \vDash ap$ iff $ap \in AP_1$.
- $\pi \vDash \varphi_1 \wedge \varphi_2$ iff $\pi \vDash \varphi_1$ and $\pi \vDash \varphi_2$.
- $\pi \vDash \neg\varphi$ iff $\pi \nvDash \varphi$.

- $\pi \vDash \bigcirc\varphi$ iff $\pi[1...n] \vDash \varphi$.
- $\pi \vDash \varphi_1 \cup \varphi_2$ iff $\exists j \geqslant 0.\ \pi[j...] \vDash \varphi_2$ and $\pi[i...] \vDash \varphi_1$ for all $0 \leqslant i < j$

The LTL model checking algorithm is described below. This algorithm uses the notion of Buchi automata [17] which is a type of ω-automaton, which extends a finite automaton to infinite inputs. It accepts an infinite input sequence if there exists a run of the automaton that visits (at least) one of the final states infinitely often.

Algorithm 1 Algorithm of LTL model checking [17]

Require:
A transition system $A = (S, s_0, Act, \Delta, AP, L)$ and LTL formula φ over AP.
Ensure:
1: Construct a Non-deterministic Buchi Automata $NBA_{\neg\varphi}$ such that $L(NBA_{\neg\varphi}) = Words(\neg\varphi)$.
2: Construct the product $A \otimes NBA_{\neg\varphi}$.
3: If there exists an accepted sequence (for example, a path Π) in $A \otimes NBA_{\neg\varphi}$, it means the model checking will return "false" with a counterexample.
4: **return** True or false.

The complexity of LTL model checking is EXPTIME with respect to the size of LTL formula. According to [17], the time complexity of LTL model checking is in $O(|TS| \times 2^{|\varphi|})$ where $|TS|$ is the size of automata and $|\varphi|$ is the size of give LTL formula. We introduce the notion of Probabilistic LTL (P-LTL). The syntax of P-LTL is same with the one of LTL. According to [41], an informal definition of P-LTL model checking is as follows: for a given finite state automaton A and a LTL formula φ, $A \vDash_{\sim pr} \varphi$, $pr \in [0, 1]$. We say A satisfies φ in a state s with probability $\sim pr$ iff every path starting from s satisfies φ with probability $\sim pr$.($\sim$ means $=, <, >, \leqslant, \geqslant$). The algorithm of P-LTL model checking is not used in this thesis. It is presented in Appendix.

2.5.2 CTL model checking

Computation tree logic [17] was birthed to overcome the shortage of LT properties which only consider linear notion of time other than branching one. Branching notion of time is a principle property of finite state automata. CTL is a temporal logic based on propositional logic with a discrete notion of time, and only future modalities. It is an important branching temporal logic that is sufficiently expressive for the formulation of an important set of system properties. The syntax of CTL model checking differs from the one of LTL with respect to branching parameters. There are two kinds of formulae for CTL: state formulae and path formulae. The grammar of state formulae is as follows:

$$\Phi \vDash true|ap|\Phi_1 \wedge \Phi_2|\neg\Phi|\forall\varphi|\exists\varphi \tag{2.2}$$

where ap is an atomic proposition and φ represents a path formulae. The grammar of path formulae shows below:

$$\varphi \vDash \bigcirc\Phi | \Phi_1 U \Phi_2 \tag{2.3}$$

Similar with LTL model checking, the semantic of CTL checking also needs some satisfaction relations. The difference is on the fact that CTL has two types formulae. Therefore there are two types of satisfaction relations: for state formulae and for path formulae. Let $ap \in AP$ be an atomic proposition, an automaton $A = (S, s_0, Act, \Delta, F, L, AP)$. The satisfaction relation for state formulae is defined as follows.

- $s \vDash ap$ iff $ap \in L(S)$.
- $s \vDash \Phi_1 \wedge \Phi_2$ iff $s \vDash \Phi_1$ and $s \vDash \Phi_2$.
- $s \vDash \neg\Phi$ iff $s \nvDash \Phi$.
- $s \vDash \exists\varphi$ iff $\pi \vDash \varphi$ for some $\pi \in Paths(s)$.
- $s \vDash \forall\varphi$ iff $\pi \vDash \varphi$ for all $\pi \in Paths(s)$.

The satisfaction relation for path formulae is defined by:

- $\pi \vDash \bigcirc\varphi$ iff $\pi[1] \vDash \varphi$.
- $\pi \vDash \varphi_1 \cup \varphi_2$ iff $\exists j \geqslant 0.\ \pi[j] \vDash \varphi_2$ and $\pi[i] \vDash \varphi_1$ for all $0 \leqslant i < j$

The basic algorithm of CTL model checking [17] is to verify a given finite state automata and CTL formulae Φ if $LTS \vDash \Phi$. The general step of CTL model checking is as follows, if $Sat(\Phi) = \{s \in S | s \vDash \Phi\}$ and $Sub(\Phi)$ is a set of sub-formulae of Φ:

- Input: a given finite state automata and CTL formulae Φ.
- Build the parse tree [2]. For Φ, divided Φ into a set of sub-formulae denoted as $Sub(\Phi)$. The atomic formulae are depicted as Equation 2.4.
- For each $\Psi_i \in Sub(\Phi)$, find $Sat(\Psi_i) = \{s \in S | s \vDash \Psi_i\}$.
- Accumulate the consequences to get $Sat(\Phi) = \bigcap(Sat(\Psi_i))$.
- Finally, we get $LTS' \vDash \Phi$ where $s \in S_{LTS'}$ also $s \in Sat(\Phi)$.

[2]The nodes of the parse tree represent the sub-formulae of Φ. The leaves stand for the constant true or an atomic proposition $a \in AP$. All inner nodes are labeled with an operator

The following equation illustrates the atomic formulae for CTL model checking.

$$\Phi \vDash true|ap|\Phi_1 \wedge \Phi_2|\neg\Phi|\exists\Box\Phi|\exists \bigcirc \Phi|\exists(\Phi_1 \cup \Phi_2) \tag{2.4}$$

The complexity of CTL model checking is in PTIME with respect to the size of CTL formula. According to [17], the time complexity of CTL model checking is in $O(|TS| \times |\Phi|)$ where $|TS|$ is the size of automata and $|\Phi|$ is the size of give CTL formula.

2.5.3 P-CTL model checking

P-CTL is CTL model checking in probabilistic system. It introduces a new operator for the CTL model checking as $P_{\sim pr}(\phi)$. Similar to CTL, the grammars of state formulae and path formulae are described as follows:

$$\Phi \vDash true|ap|\Phi_1 \wedge \Phi_2|\neg\Phi|P_{\sim pr}(\phi) \tag{2.5}$$

where $P_{\sim pr}(\phi$ means ϕ is true with probability pr, $pr \subseteq [0,1]$, $\sim \in \{<, >, \leqslant, \geqslant\}$.

$$\varphi \vDash \bigcirc\Phi|\Phi_1 \cup \Phi_2|\Phi_1 \cup^{\leqslant n} \Phi_2 \tag{2.6}$$

For the non-probabilistic operators, the method of P-CTL model checking is similar with the one of CTL in spite of considering quantitative results. As to the probabilistic operator P, it needs to compute the probability $Prob(s, \phi)$ for every state such that $Sat(P_{\sim pr}(\phi)) = \{s \in S | Prob(s, \phi) \sim pr\}$, $Prob(s, \phi)$ denoting the probability of s which satisfies ϕ.

Chapter 3

Model description

This chapter describes the notion of probabilistic data-aware business process model (pd-processes in short) in detail. The semantic of pd-processes is introduced in terms of possible execution trees and the notion of world-partition automata is proposed to help us to verify pd-process model in the next chapter.

3.1 Probabilistic data-aware business process model

In this section, we continue using the example of car insurance company to illustrate our probabilistic data-aware business process framework. Roughly speaking, the business process calculates insurance rates using three different procedures depending on risk and profit factors: (i) if the associated risk is considered low, then a premium is calculated and either a quote is automatically elaborated and proposed to the customer (if the forecast profit is at least medium) or submitted for approval to a human expert, or (ii) if the associated risk is considered medium, then a premium is calculated and either a quote is automatically elaborated and proposed to the customer (if the forecast profit is high) or submitted for approval to a human expert, (iii) if the associated risk is considered as high, in this case an advanced approval procedure is required. Insurance companies usually try to improve their risk management by using probabilistic risk analysis and profit forecasts. For example, the probabilistic database depicted at Table 2.3 could be used by the business process of figure 3.1 to provide better estimates for premiums.

We present below a formal definition of the notion of probabilistic data-aware business processes (pd-processes).

Definition 3.1 (probabilistic data-aware process)**.** A probabilistic data-aware process (pd-process) is a tuple $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$, where:

- S is a finite set of states, with $s_0 \in S$, the starting state.
- D is a probabilistic database with possible worlds $\mathcal{W}(D)$.
- Act is a finite set of actions or activities.
- G is a finite set of guards defined as boolean queries over the database D.
- $\Delta \subseteq S \times Act \times G \times S$, the transition relations, is a set of guarded transitions.
- $F \subseteq S$ is the set of final states.
- AP is a set of atomic propositions.
- L is a labelling function $L : S \to 2^{AP}$.

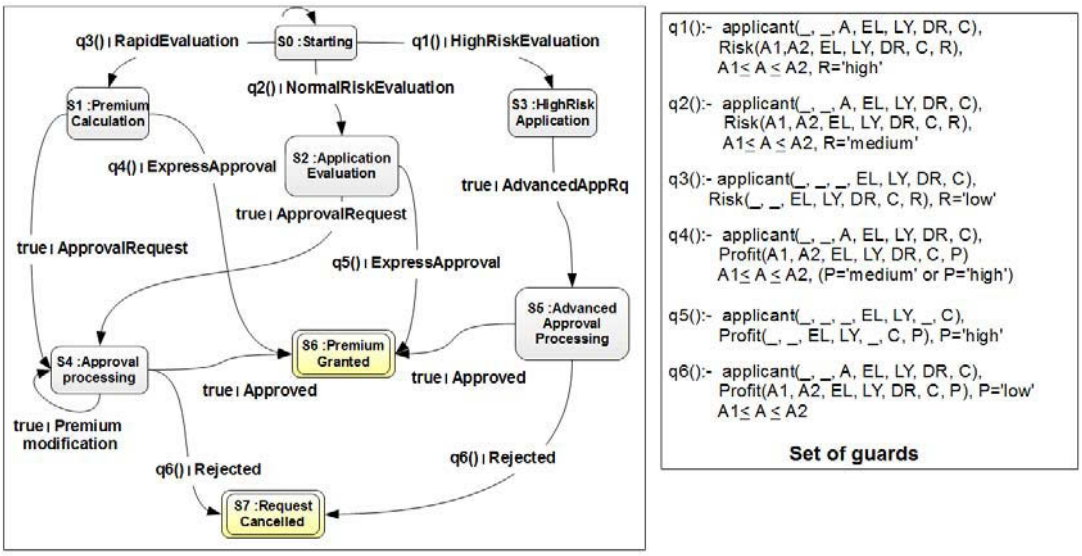

FIGURE 3.1: PremCalc: an insurance premiums calculation business process.

Figure 3.1 shows a graphical representation of a premiums calculation pd-process, called PremCalc. This process uses the insurance probabilistic database D_{ins} given at Table 2.2. At the beginning, the PremCalc process is at the initial state Starting. Then, depending on the risk level associated with the current applicant, PremCalc moves to one of the states Premium Calculation, Application Evaluation or High Risk Application. This conditional move is specified using transitions guards expressed as boolean queries over the probabilistic database D_{ins}. For example, the transition labelled 'q_3 | Rapid Evaluation' from state Starting to state Premium Calculation specifies that when the PremCalc process is at state Starting and the guard q_3 is true, then PremCalc may execute the activity Rapid Evaluation and moves to state Premium Calculation. At this stage, two observations are worth to mention. First, note that pd-processes are non-deterministic processes as it can be observed in the example where the two transitions outgoing from

state Premium Calculation have non exclusive guards. This means that the applications that are at state Premium Calculation and which satisfy the guard q_4 may be either processed automatically, i.e., a quote is automatically elaborated and sent to the customer without requiring any approval (transition Express Approaval) or may require a manual approval (transition Approval Request). Second, the presence of probabilities in the database makes process executions probabilistic. For example, the execution of the transitions 'q_3 | Rapid Evaluation' is determined by the probability of its guard q_3 (i.e., the probability to have q_3 evaluated to true). Hence, the pd-process PremCalc can be viewed as a probabilistic process with probabilities associated with transitions. As discussed below (c.f., notion of possible execution trees), probabilities of transitions determine the branching choices available during a given process execution. Moreover, it should be noted that probabilities of transitions are not independent. Arbitrary and complex correlations between transitions probabilities may arise depending on the considered probabilistic database and on the connections that exist between transitions guards (e.g., disjoint guards, containment, overlapping, ...).

To illustrate verification method in latter chapters clearly, a naive example is provide as follows. Figure 3.2(a) shows another variant of a premium calculation business process, called NormCalc. This process evaluates customers' applications using a simpler procedure which starts by launching the activity NormalRiskEvaluation. Then, two cases are possible to terminate an execution: (i) for an applicant whose age and city belong to a category where the forecast profit is reasonable (c.f., guard q_2'), an express approval is authorized (ii) for an applicant whose driving record and city belong to a category with an *acceptable* level of risk (c.f., guard q_1'), a manual approval is then requested. Figure 3.2(b) shows the set of all possible execution trees of the process NormCalc.

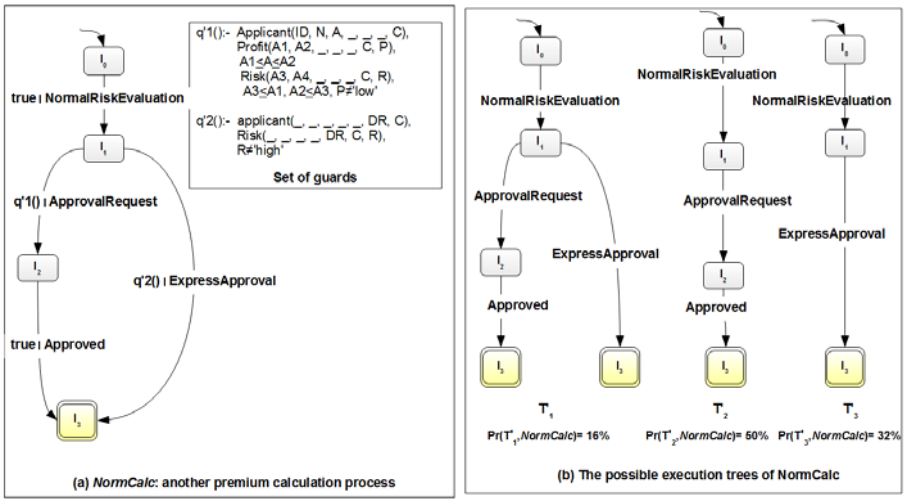

FIGURE 3.2: Another insurance premium calculation business process.

Semantic

Various classes of process semantics have been studied in the literature [42]. A line of demarcation between existing semantics lies in the distinction between *linear time* and *branching time* semantics. When processes are compared with respect to *branching time* semantics execution paths as well as branching structures of processes must be taken into account. Usually, *simulation preorder* is used in the literature as a relation to compare processes with respect to their branching structures [42]. Following branching time semantics, possible executions allowed by a process are characterized in terms of trees, called *execution trees*, instead of paths. In a nutshell, execution trees of a process A capture all the executions paths of A as well as the branching structures of A. In the case of pd-processes, execution trees depend on the evaluation of guards which is determined by the considered possible world of the probabilistic database.

As an example, Figure 3.3 shows some execution trees of the process PremCalc in the possible worlds W_{256} and W_2. The presented trees are *complete* execution trees since their leaves are made of final states. Hence, each branch in these trees that starts from the root and ends at a leaf forms an execution path of the process PremCalc in the considered world. For instance, the sequence of activities NormalRiskEvaluation.ApprovalRequest.Premium Modification.Approved, which appears as a branch in the execution tree T_3, is a possible execution path of the process PremCalc in the world W_{256}. In addition, an execution tree captures the branching structures of a process. For example, tree T_3 shows that after the execution of the activity RapidEvaluation, the process PremCalc will have a choice to either execute the activity ExpressApproval or the activity ApprovalRequest.

To formally define the notions of execution trees, we use the following definition of a tree: A tree is a set $\tau \subseteq \mathbb{N}^*$ such that if $xn \in \tau$, for $x \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $x \in \tau$ and $xm \in \tau$ for all $0 \leq m < n$. The elements of τ represent nodes: the empty word ϵ is the root of τ, and for each node x, the nodes of the form xn, for $n \in \mathbb{N}$, are children of x. Given a pair of sets L and M, an $\langle L, M\rangle$-labelled tree is a triple (τ, λ, δ), where τ is a tree, $\lambda : \tau \to L$ is a node labelling function that maps each node of τ to an element in L, and $\delta : \tau \times \tau \to M$ is an edge labelling function that maps each edge (x, xn) of τ to an element in M. Then, every path $\rho = \epsilon, n_0, n_0n_1, \ldots$ of τ generates a sequence $\Gamma(\rho) = \lambda(\epsilon).\delta(\epsilon, n_0).\lambda(n_0).\delta(n_0, n_0n_1).\lambda(n_0n_1)\ldots$. of alternating labels from L and M.

Informally, if L and M correspond to the sets of states S and actions Act of a pd-process A, then we can use an $\langle S, Act\rangle$-labeled tree to characterize the semantics of A. In particular, the branches of the tree (once mapped with the labeling functions) represent execution paths, and the tree hierarchy reflects the branching structures of the process.

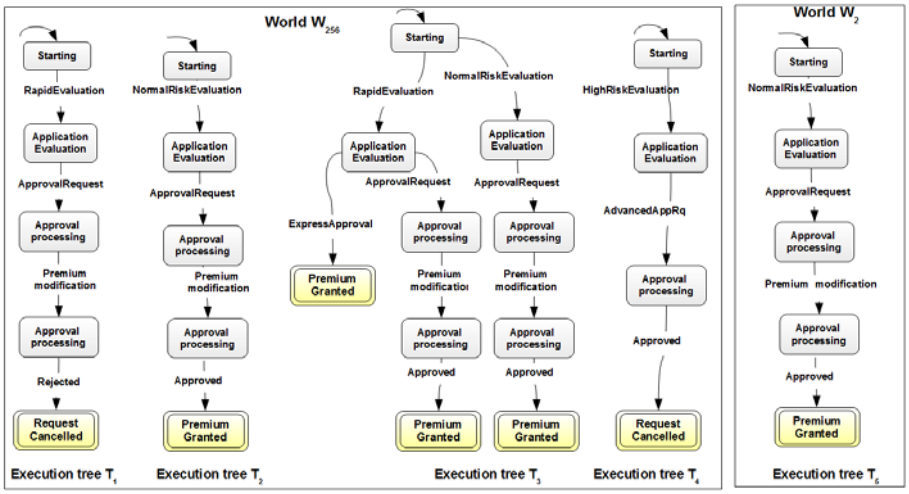

FIGURE 3.3: Examples of execution trees of the worlds W_{256} and W_2.

Definition 3.2 (Execution trees)**.** Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ be a pd-process. An execution tree of A in a world $W \in \mathcal{W}(D)$ is a $\langle S, Act\rangle$-labeled tree $T = (\tau, \lambda, \delta)$ such that:

(i) $\lambda(\epsilon) = s_0$ and for every leaf $x \in \tau$ we have $\lambda(x) \in F$, and

(ii) for each edge (x, xn) of τ, there exists a guard $g \in G$ such that $(\lambda(x), \delta(x, xn), g, \lambda(xn)) \in \Delta$ and $g(W) = True$.

We denote by $Tr(A, W)$ the set of execution trees of A in the world W.

The set of execution trees of a given pd-process may be infinite. Continuing with the example, due to the presence of a loop Premium Modification at the state Approval processing of the process PremCalc, the set $Tr(\mathsf{PremCalc}, W_{256})$ of its execution trees in the world W_{256} is infinite. For example, the execution tree T_3 of Figure 3.3 includes a one time execution of this loop. Starting from T_3, an infinite number of execution trees may be constructed by increasing the number of times this loop is executed.

Definition 3.3 (Possible execution trees)**.** Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ be a pd-process.

A $\langle S, Act\rangle$-labelled tree $T = (\tau, \lambda, \delta)$ is a **possible execution tree** of A iff $\exists W \in \mathcal{W}(D)$ and $\exists \lambda' : \tau \to S$ such that $T' = (\tau, \lambda', \delta) \in Tr(A, W)$.

The probability of a possible execution tree T of A is: $Pr(T, A) = \sum_{\substack{W \in \mathcal{W}(D) \\ T' \in Tr(A,W)}} Pr(W)$.

We denote by $Tr(A)$ the set of all possible execution trees of a process A.

A possible execution tree T is simply an execution tree augmented with the probability of occurrence of T. Note that two execution trees are considered *equal* if they differ only w.r.t. the labels of their states (e.g., T is equal to T' in definition 3.3). Hence, a probability of a possible execution tree T is calculated as the sum of the probabilities of the possible worlds to which T, modulo renaming of states, belongs. As an example, probabilities of the execution trees $\mathsf{T_1}$ and $\mathsf{T_2}$ of PremCalc depicted at Figure 3.3 are: $\mathsf{Pr}(\mathsf{T_1}, \mathsf{PremCalc}) = 24\%$ and $\mathsf{Pr}(\mathsf{T_2}, \mathsf{PremCalc}) = 80\%$.

Possible execution trees show the intricate relations between probabilities of transitions and branching choices. Continuing with the example, the possible execution trees $\mathsf{T_1}$, $\mathsf{T_2}$ and $\mathsf{T_3}$ reveal that when the PremCalc process is at the state Starting we may have the following situations:

(i) the only possible choice is to execute the transition RapidEvaluation (tree $\mathsf{T_1}$ with a probability $Pr(T_1, \mathsf{PremCalc}) = 24\%$), or (ii) the only possible choice is to execute the transition NormalRiskEvaluation (tree $\mathsf{T_2}$ with a probability $Pr(T_2, \mathsf{PremCalc}) = 80\%$), or (iii) there is a choice to execute either transition NormalRiskEvaluation or transition RapidEvaluation (tree $\mathsf{T_3}$ with a probability $Pr(T_3, \mathsf{PremCalc}) = 16\%$). Such a complex correlation between probabilities of transitions and branching choices makes any structural analysis of pd-processes a difficult task. As explained below, one consequence is that it is not straightforward to provide a simple definition of simulation based only on the structures of the processes. It is worth noting that although the trees $\mathsf{T_1}$ and $\mathsf{T_2}$ are both *sub-trees* of $\mathsf{T_3}$, they both have a probability greater than the one of $\mathsf{T_3}$. This is because $\mathsf{T_1}$ and $\mathsf{T_2}$ belong to more possible worlds than $\mathsf{T_3}$. This general property, which states that the probability of a tree T is always less or equal to the probability of its sub-trees, will be exploited later to devise a simulation algorithm for pd-processes.

3.2 World-partition automata

For the reason of simulation relation test and model checking, we describe below a refinement of a pd-process structure into a set of automata, called *world-partition automata*, that can be used to *structurally* characterize simulation and facilitate the verification method of pd-processes. We recall that for a set G, the set 2^G denotes the power set of G (i.e., the set of all subsets of G).

Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ be a pd-process with $G = \{q_1, \ldots, q_n\}$ a set of boolean queries used as guards of transitions in A. Let P_G be a set of boolean queries

Process	Partitions	Associated query	P. worlds	Probability
PremCalc	P_1	$q_{P_1} = \neg q_1 \wedge q_2 \wedge \neg q_3 \wedge \neg q_4 \wedge q_5 \wedge q_6$	$\{W_2, ...\}$	6.048%
	P_2	$q_{P_2} = q_1 \wedge q_2 \wedge \neg q_3 \wedge q_4 \wedge q_5 \wedge \neg q_6$	$\{W_3, ...\}$	2.592%
	P_3	$q_{P_3} = \neg q_1 \wedge q_2 \wedge \neg q_3 \wedge q_4 \wedge q_5 \wedge q_6$	$\{W_4, ...\}$	6.048%
	...	...	...	
NormCalc	P'_1	$q_{P'_1} = q'_1 \wedge q'_2$	$\{W_4, ...\}$	16%
	P'_2	$q_{P'_2} = q'_1 \wedge \neg q'_2$	$\{W_5, ...\}$	34%
	P'_3	$q_{P'_3} = \neg q'_1 \wedge q'_2$	$\{W_2, ...\}$	16%
	P'_4	$q_{P'_4} = \neg q_1 \wedge \neg q_2$	$\{W_6, ...\}$	34%

TABLE 3.1: Example partitions

obtained as follows:
(i) $\forall P \in 2^G$, $q_P := (\bigwedge_{q \in P} q) \wedge (\bigwedge_{q' \notin P} \neg q')$, and
(ii) $P_G = \{q_P \mid P \in 2^G\}$.

Note that, the set P_G forms a partition of the possible worlds of the database D in the sense given by the following lemma.

Lemma 3.4. *Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ and let P_G be the set of guards constructed as explained above. Then, $\forall W \in \mathcal{W}(D)$, there exists a unique $q_P \in P_G$ such that $q_P(W) = true$.*

The proof of this lemma is straightforward since, by construction of P_G, $\forall W \in \mathcal{W}(D)$ we have: (i) $\bigvee_{q_P \in P_G} q_P(W) = true$, and (ii) $\forall q_{pi}, q_{pj} \in P_G$, with $i \neq j$, then $q_{pi}(W) \wedge q_{pj}(W) = false$. Hence, each boolean query $q_P \in P_G$ identifies a unique subset of $\mathcal{W}(D)$ (i.e., the set $\{W \in \mathcal{W}(D) \mid q_P(W) = true\}$).

In the sequel, we use the term *partition* q_P to refer to the subset of $\mathcal{W}(D)$ identified by q_P. Table 3.1 shows examples of partitions related to the set of guards of the processes PremCalc and NormCalc. For each partition, an associated probability is computed (c.f., last column in Table 3.1) using a probabilistic database management system. Also note that it may happen that a probability associated to a given partition is equal to *zero*. In this case, the corresponding query is removed from the set P_G.

We introduce below the notion of *world-partition automata* as a mean to split the behaviour described by a given pd-process w.r.t. the possible worlds of the underlying probabilistic database. More precisely, the goal is to split the set of possible execution trees of a pd-process A into subsets of trees each of which is described by a distinct unguarded automaton.

Definition 3.5 (World-partition automata)**.** Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ be a pd-process and let $P_G = \{q_{P_1}, \ldots, q_{P_n}\}$ defined as previously. A world-partition automata of A using P_G is a set of automata $A_{P_G} = \{A_{P_1}, \ldots, A_{P_n}\}$, where, $\forall q_{Pi} \in P_G$, a

corresponding automaton $A_{P_i} = (S, s_0, D_{P_i}, Act, G_{P_i}, \Delta_{P_i}, F, AP, L)$ is constructed from A as follows:

(i) the components S, s_0, Act, and F, remain unchanged,

(ii) the set of guards is: $G_{P_i} = \{true\}$ and the database is $D_{P_i} = \emptyset$,

(iii) the set of transitions is: $\Delta_{P_i} = \{(s, a, true, s') \mid (s, a, g, s') \in \Delta \text{ and } g \in P_i\}$.

The probability function Pr is extended to world-partition automaton as follows: $\forall A_P \in A_{P_G}$, then $Pr(A_P) = \sum_{\substack{W \in \mathcal{W}(D) \\ q_P(W) = true}} Pr(W)$

Hence, an automaton $A_{P_i} \in A_{P_G}$ is simply a copy of the process A from which are removed the transitions having a guard g satisfying the condition $Pr(g \wedge q_{P_i}) = 0$ (or equivalently, $\forall W \in \mathcal{W}(D) \mid g \wedge q_{P_i}(W) = \mathsf{false}$)[1]. Note that such a test can be achieved easily by checking whether $g \in P_i$ (since we have: $Pr(g \wedge q_{P_i}) = 0$ iff $g \notin P_i$). From item (ii) of this definition, each automaton $A_{P_i} \in A_{P_G}$ is an unguarded automaton (i.e., all its guards are set to true).

Figure 3.4(a) shows the world-partition automata of the process NormCalc while the Figure 3.4(c) shows two automata from the world-partition automata of the process PremCalc.

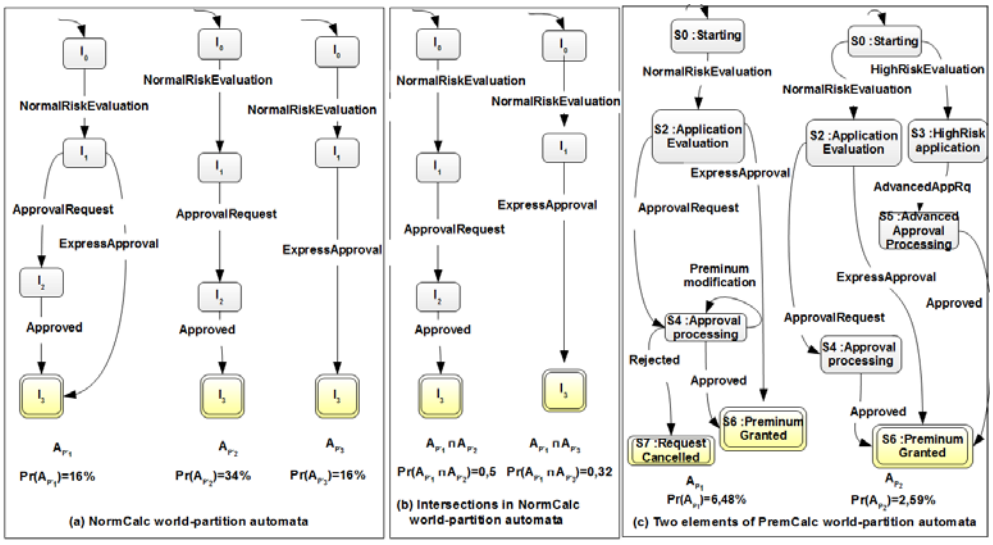

FIGURE 3.4: Example of world-partition automata.

Intuitively, a world-partition automaton $A_P \in A_{P_G}$ describes the behaviour of A in all the possible worlds belonging to the partition q_P. The following lemma makes explicit

[1]Indeed, A_{P_i} must then be *cleaned* by removing states that do not belong to a direct path from the initial state to a final state. Such a cleaning can be achieved in linear time using algorithms such as breadth first search.

the connection between the behaviour of a pd-process A and the behaviours described by its world-partition automata.

Lemma 3.6. *Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ be a pd-process and let A_{P_G} its set of world-partition automata. Then:*
(i) let $W \in \mathcal{W}(D)$ be a possible world of D that belongs to a partition $q_P \in A_{P_G}$. Then, T is an execution tree of A in the world W iff $T \in Tr(A_P)$.

(ii) $T \in Tr(A)$ with $Pr(T, A) > 0$ iff $\exists\{A_{P_{i_1}}, \ldots, A_{P_{i_l}}\} \subseteq A_{P_G}$ such that: $T \in Tr(A_{P_{ij}})$, $\forall i_j \in \{i_1, \ldots, i_l\}$, and $Pr(T, A) = \sum_{i_j \in \{i_1,\ldots,i_l\}} Pr(A_{P_{i_j}})$

Proof.

- (i) ($\Rightarrow$) Let $W \in \mathcal{W}(D)$ and $q_P \in A_{P_G} \mid q_P(W) = \mathsf{true}$ and let $T = (\tau, \lambda, \delta) \in Tr(A, W)$.
  By definition 3.2, $\forall (x, xn) \in \tau$, $\exists g \in G$ such that $(\lambda(x), \delta(x, xn), g, \lambda(xn)) \in \Delta$ and $g(W) = True$. Let G_T be the set containing such guards g.
  From lemma 3.4,we have $G_T \subseteq P$
  Hence, $T \in Tr(A_P, W)$, by construction of A_P.

- (i)($\Leftarrow$) Straightforward.

- (ii)($\Rightarrow$) Assume $T \in Tree(A)$ with $Pr(T, A) > 0$. Hence, from definition 3.3, there exists $\{W_{i_1}, \ldots, W_{i_l}\} \subseteq \mathcal{W}(D) \mid T \in Tr(A, W_{i_j})$ and $Pr(T, A) = \sum_{i_j \in \{i_1,\ldots,i_l\}} Pr(W_{i_j})$.
  From (1), we can derive that $\exists\{A_{P_{i_1}}, \ldots, A_{P_{i_l}}\} \subseteq A_{P_G}$ such that $T \in Tr(A_{P_{i_j}})$, $\forall i_j \in \{i_1, \ldots, i_l\}$, and $Pr(T, A) = \sum_{i_j \in \{i_1,\ldots,i_l\}} Pr(A_{P_{i_j}})$.

- (ii)($\Leftarrow$) Assume that $\exists\{A_{P_{i_1}}, \ldots, A_{P_{i_l}}\} \subseteq A_{P_G}$ such that: $T \in Tr(A_{P_{i_j}})$, $\forall i_j \in \{i_1, \ldots, i_l\}$. Hence, $T \in Tr(A)$ and $Pr(T, A) > 0$ (from (1) and definition 3.3).

□

Chapter 4

Verification methods

This chapter mainly focuses on the method of testing simulation relation and model checking in the context of pd-processes. In the first section, with the help of world partition automata introduced in the previous chapter, an algorithm of testing simulation relation of pd-processes is given and the complexity is in 2-EXPTIME. In the second part, P-LTL and P-CTL model checking algorithms over pd-processes are studied as well as their complexity.

4.1 Simulation preorder

The notion of *simulation* is used in the literature to compare finite state machine with respect to their branching structures [42]. Usually simulation is defined as a relation between the states of the considered processes. In the case of pd-processes, and due to the tight connection between possible execution trees and possible worlds of a probabilistic database, it is not easy to provide such a structural definition (i.e., as a general relation between states). This is because, whether or not a given state s is simulated by another state s' depends on the considered possible world. As a consequence, instead of a structural definition, we provide below a definition of simulation based on the semantics of pd-processes.

Definition 4.1 (Simulation relation between pd-processes)**.** Let $A = (S, s_0, D, Act, G, \Delta, F)$ and $A' = (S', s'_0, D', Act', G', \Delta', F')$ be two pd-processes. Then, A is simulated by A', noted $A \lesssim A'$, iff: $\forall T = (\tau, \lambda, \delta) \in Tr(A)$, $\exists \lambda' : \tau \to S'$ such that $T' = (\tau, \lambda', \delta) \in Tr(A')$, and $Pr(T, A) \leq Pr(T', A')$.

Hence, semantics of simulation is defined as a containment between the sets of possible execution trees of the considered pd-processes. According to this definition, when

comparing execution trees of two processes, the labels of the activities are taken into account while names of the states are meaningless. Hence, if a process A is simulated by a process A', then every possible execution tree of A is also a possible execution tree of A' (modulo renaming of states) with an equal or higher probability. It is worth noting that the provided definition of simulation is conservative in the sense that when it is applied to pd-processes with non-probabilistic databases (i.e., having probability of each tuple equal to 1), it matches non-probabilistic simulation on conventional LTSs. If we come back to the example of NormCalc and PremCalc, there are numerous situations where it is interesting to identify whether the process NormCalc is a refinement of the process PremCalc w.r.t. to the simulation preorder. Unfortunately, definition 4.1 is *semantic* (i.e., it defines the meaning of simulation as a relation between possible execution trees) and not structural (i.e., does not define a relation between states and transitions of the processes). Therefore, there is no direct way to derive a simulation algorithm from such a definition (since testing inclusion between potentially infinite sets of possible execution trees is not feasible). As a consequence, we can use the notion of partition automata to decompose a pd-process into a set of (unguarded) automata that can be analyzed separately to structurally characterize simulation relation between pd-processes.

While a world-partition automata A_{P_G} enables to split an original process A into a set of automata that describe all the possible execution trees of A, it is still not easy to reason separately on elements of A_{P_G} to test simulation. This is due to the fact that a probability of a possible tree T of A may be obtained from a subset of A_{P_G} (and not only from a unique element of A_{P_G}) (c.f., lemma 3.6). Such problematic execution trees belong to intersections of elements of A_{P_G}. Figure 3.4(b) shows the (non-empty) automata obtained from intersections between subsets of the world-partition automata of the process NormCalc[1]. Then it is easy to see that execution trees of the automaton $A_{P'_1} \sqcap A_{P'_2}$ are possible execution trees of the process NormCalc with a probability equal to $Pr(A_{P'_1}) + Pr(A_{P'_2})$. Therefore, to characterize precisely the probabilities of every possible execution tree by a unique automata, there is a need to compute the closure of world-partition automata w.r.t. the intersection operation. In the example, the world-partition automata of NormCalc needs to be augmented with the automata of Figure 3.4(b) in order to obtain the closure of the world-partition automata of NormCalc w.r.t. intersection operation. The closure of world-partition automata, called closure-automata, is formally defined below after the introduction of some needed notation. Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ be a pd-process and let A_{P_G} its corresponding world-partition automata. For a set $\psi \in 2^{A_{P_G}}$ (i.e., a subset of A_{P_G}), we define $A_{\sqcap_\psi} := \prod_{A_P \in \psi} A_P$. Therefore, $A_{\sqcap_\psi}$ is an unguarded automata which describes the behavior common to all the automata of the set ψ.

[1]The symbol $\sqcap$ denotes conventional intersection between LTSs.

Definition 4.2. (closure-automata) Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ a pd-process and let A_{P_G} its corresponding world-partition automata. The closure of the world-partition automata of A is given by the set $C_L(A_{P_G}) = \{A_{\sqcap_\psi} \mid \psi \in 2^{A_{P_G}}\}$, where each transition system $A_{\sqcap_\psi} \in C_L(A)$ is associated with a probability distribution $Pr(A_{\sqcap_\psi}) = \sum_{A_P \in \psi} Pr(A_P)$.

As an example, the set $C_L(\mathsf{NormCalc})$, corresponding to the closure of world-partition automata of the process NormCalc, is made of the automata of Figures 3.4(a) and (b) and their associated probabilities (shown in the figures).

Simulation test algorithm

This part studies the algorithm of testing simulation relation of pd-processes. As a main technical result of this chapter, the next theorem provides a structural characterization of the simulation relation between two pd-processes.

Theorem 4.3. *Let A and B be two pd-processes. Let $C_L(A_{P_G})$ be the closure-automata of A and let $B_{P_{G'}}$ be the world-partition automata of B. Then:*
$A \lesssim B$ ***iff*** $\forall A_{\sqcap_\psi} \in C_L(A_{P_G})$, $Pr(A_{\sqcap_\psi}) \leq \sum_{\substack{B_{P'} \in B_{P_{G'}} \\ A_{\sqcap_\psi} \lesssim B_{P'}}} Pr(B_{P'})$

sketch. ($\Leftarrow$) Assume that $\forall A_{\sqcap_\psi} \in C_L(A_{P_G})$, we have $Pr(A_{\sqcap_\psi}) \leq \sum_{\substack{B_{P'} \in B_{P_{G'}} \\ A_{\sqcap_\psi} \lesssim B_{P'}}} Pr(B_{P'})$ (i).

Let $T = (\tau, \lambda, \delta) \in Tr(A)$ with $Pr(T, A) > 0$. By lemma 3.6, $\exists\{A_{P_{i_1}}, \ldots, A_{P_{i_l}}\} \subseteq A_{P_G}$ such that: $T \in Tr(A_{P_{i_j}})$, $\forall j \in \{1, \ldots, l\}$, and $Pr(T, A) = \sum_{i_j \in \{i_1, \ldots, i_l\}} Pr(A_{P_{i_j}})$.
Let $A_{\sqcap_\psi} := A_{P_{i_1}} \sqcap \ldots \sqcap A_{P_{i_l}}$. From assumption (i), we derive: $\exists\{B_{P'_{i_1}}, \ldots, B_{P'_{i_k}}\} \subseteq B_{P'_G}$ such that $A_{\sqcap_\psi} \lesssim B_{P'_{i_j}}$, $j \in [1, k]$, and $Pr(_{\sqcap_\psi}) \leq \sum_{j \in [1,k]} Pr(B_{P'_{i_j}})$. Hence, $\forall B_{P'_{i_j}}$, with $j \in [1, k]$, $\exists \lambda' : \tau \to S'$ such that $T' = (\tau, \lambda', \delta) \in Tr(B_{P'_{i_j}})$. From lemma 3.6, $T' \in Tr(B)$ and $Pr(T', B) \geq \sum_{j \in [1,k]} Pr(B_{P'_{i_j}}) \geq Pr(T, A)$. Hence, $A \lesssim B$ (by definition 4.1).

($\Rightarrow$) Assume that $A \lesssim B$ and $\exists A_{\sqcap_\psi} \in C_L(A_{P_G})$ such that

$Pr(A_{\sqcap_\psi}) \geq \sum_{\substack{B_{P'} \in B_{P_{G'}} \\ A_{\sqcap_\psi} \lesssim B_{P'}}} Pr(B_{P'})$.

In this case, the *maximal* execution tree of $A_{\sqcap_\psi}$ does not have any corresponding tree in B with enough probability.
Hence, $A \not\lesssim B$ (which contradict the assumption)

□

Continuing with the example, checking whether the pd-process NormCalc is simulated by PremCalc can be achieved by checking whether for each automaton $A_{\sqcap_\psi}$ in $C_L(\mathsf{NormCalc})$, there is a subset of automata in the world-partition automata of PremCalc each of wich simulates $A_{\sqcap_\psi}$ and having a sum of probabilities greater or equal to the probability of $A_{\sqcap_\psi}$. We recall that, using the automata depicted at Figure 3.4, the closure of world-automata of NormCalc is $C_L(\mathsf{NormCalc}) = \{A_{P'_1}, A_{P'_2}, A_{P'_3}, A_{P'_1} \sqcap A_{P'_2}, A_{P'_1} \sqcap A_{P'_3}\}$. Consider the case of the automaton $A_{P'_1}$ of this set. Using any state of the art simulation algorithm on unguarded LTSs, one can check that this automaton is respectively simulated by the automaton A_{P_1} and by the automaton A_{P_2} of Figure 3.4(c). In fact, A_{P_1} and A_{P_2} are just two examples of a family of automata from the world-partition automata of PremCalc that simulates $A_{P'_1}$. Indeed, any automaton of the world-partition automata of PremCalc that belongs to a partition satisfying the guards q_2 and q_5 of, respectively, transitions NormalRiskEvaluation and ExpressEvaluation of PremCalc simulates $A_{P'_1}$. The total number of such partitions is 2^4 with the sum of the associated probabilities equal to 48% (which is greater than $Pr(A_{P'_1})$). Similar reasoning can be extended to all other elements of $C_L(\mathsf{NormCalc})$ to show that conditions of theorem 4.3 are satisfied and hence NormCalc $\lesssim$ PremCalc.

Complexity analysis. Let $A = (S, s_0, D_A, Act, G, \Delta, F)$ be a pd-process. We use $|X|$ to denote the cardinality of a set X. We extend this notation to pd-processes and we write $|A|$ to denote the size of the process A defined in terms of its total number of guards, transitions and states (i.e., $|A| = |S| + |\Delta| + |G|$). We use also the notation $|D_A|$ to denote the size of the probabilistic database used by A defined in terms of total number of tuples in D. We study the complexity of the problem of checking simulation between two pd-processes A and B w.r.t. two dimensions: (i) *expression complexity*, which assumes that $|D_A|+|D_B|$ is fixed while $|A|+|B|$ is variable, and (ii) *data complexity*, which assumes that $|A| + |B|$ is fixed while $|D_A| + |D_B|$ is variable.

Theorem 4.4. *Let A and B be two pd-processes. The problem of checking whether $A \lesssim B$ is:*
(i) in $\mathcal{O}(f(|D|))$ in data complexity, where $f(|D|)$ is the data-complexity of computing the probabilities of a boolean query on a probabilistic database D,
(ii) EXPTIME*-hard w.r.t. the expression complexity,*
(iii) can be solved in 2-EXPTIME *w.r.t. $|A| + |B|$.*

Therefore, checking simulation between pd-processes is intractable w.r.t. the size of the LTSs while, interestingly, it does not introduce additional overhead w.r.t. to probabilistic query evaluation in data-complexity. We refer to [8] for detailed results regarding the complexity of this latter problem (i.e., complexity of function $f(|D|)$) in the context of disjoint-independent databases. This result is encouraging because data-complexity is

the most significant factor in our context. Indeed, the size of the database of a pd-process can be expected to be several order of magnitude higher than the size of its LTS.

Proof. of theorem 4.4. We give the proofs in the following order: (iii), (i) and (ii). Let $A = (S, s_0, D_A, Act, G, \Delta, F)$ and $B = (S', s'_0, D_B, Act', G', \Delta', F')$ be two pd-processes.
Proof of (iii) According to theorem 4.3, checking whether $A \lesssim B$, can be achieved in three steps:

1. Computing world-partition automata of A_{P_G} and $B_{P'_G}$. The size of A_{P_G} is bounded by $2^{|G|}$ and, from definition 3.5, one can derive that each automata in A_{P_G} has a size bounded by $|A|$ and can be constructed: (i) in time linear in $|A|$, and (ii) by evaluating one probabilistic query q. A similar reasoning applies for $B_{P'_G}$.

2. Computing the closure $C_L(A_{P_G})$ of the world-partition automata of A. The size of $C_L(A_{P_G})$ is bounded by $2^{|A_{P_G}|} \leq 2^{2^{|G|}}$. The size of each automata of $C_L(A_{P_G})$ is bounded by $2^{(log(|A|)*2^{|G|})}$ (corresponding to the largest intersection of elements of A_{P_G} whose size is bounded by $|A|^{2^{|G|}}$). Hence, computing the closure $C_L(A_{P_G})$ is bounded by $2^{2^{|G|}} \times 2^{(log(|A|)*2^{|G|})} = 2^{2^{|G|}*(log(|A|+1))}$.

3. Checking whether condition of theorem 4.3 is satisfied which amounts mainly to testing simulation between pairs in $C_L(A_{P_G}) \times B_{P'_G}$, a set bounded by $2^{2^{|G|}} \times 2^{|G|}$. Due to the size of automata in $C_L(A_{P_G})$ and $B_{P'_G}$, the simulation test is in $O(2^{2^{|G|}})$. Hence, this step can be achieved in $O(2^{2^{|G|}})$.

Therefore, if we omit the cost of evaluating probabilistic queries, checking whether $A \lesssim B$ can be achieved in 2-EXPTIME in $|A| + |B|$.
Proof of (i) From proof of (iii), step 2, the number of queries to be evaluated over the probabilistic database is bounded by $2^{|G|+|G'|}$ (computation of A_{P_G} and $B_{P'_G}$). Hence, assuming $|A| + |B|$ to be a constant enables to check whether $A \lesssim B$ with a data-complexity that matches the complexity of probabilistic query evaluation.
Proof of (ii) We give now the proof of hardness. We show that checking whether $A \lesssim B$ is EXPTIME-hard, using a reduction from the following problem:
Input: $P_1, P_2, ...P_n, P$, a set of (unguarded) automata.
Problem: does $P_1 \times \ldots \times P_n \lesssim P$?

This problem is known to be EXPTIME-hard even when the $P_i s$ automata have disjoint alphabets [43]. We first reformulate this problem using intersection instead of shuffle then we present the reduction to pd-processes simulation.

Let $P_i = (S_{P_i}, s_0^i, Act_{P_i}, \Delta_{P_i}, F_{P_i})$, for $i \in [1,n]$, and $P = (S_P, s_0^P, Act_P, \Delta_P, F_P)$. We assume that $Act_{P_i} \cap Act_{P_j} = \emptyset$, $\forall i,j \in [1,n]$ with $i \neq j$.

For each $P_i = (S_{P_i}, s_0^{P_i}, Act_{P_i}, \Delta_{P_i}, F_{P_i})$, with $i \in [1,n]$, we construct an automata $\tilde{P}_i = (\tilde{S}_{P_i}, \tilde{s}_0^{P_i}, \tilde{Act}_{P_i}, \tilde{\Delta}_{P_i}, \tilde{F}_{P_i})$ as follows:

- $\tilde{S}_{P_i} = S_{P_i}$, $\tilde{s}_0^{P_i} = s_0^{P_i}$ and $\tilde{F}_{P_i} = F_{P_i}$,
- $\tilde{Act}_{P_i} = \bigcup_{i=1}^{n} Act_{P_i}$,
- $\tilde{\Delta}_{P_i} = \Delta_{P_i} \cup \{(s,a,s) \mid s \in \tilde{S_{P_i}} \text{ and } a \in \tilde{Act}_{P_i} \setminus Act_{P_i}\}$

The following lemma extends to simulation equivalence a result already known in the case of language equivalence.

Lemma 4.5. *Let P_i and $\tilde{P}_i$, with $i \in [1,n]$, be two sets of automata defined as described above. Then: $P_1 \times \ldots \times P_n \lesssim \tilde{P}_1 \sqcap \ldots \sqcap \tilde{P}_n$ and $\tilde{P}_1 \sqcap \ldots \sqcap \tilde{P}_n \lesssim P_1 \times \ldots \times P_n$*

Proof. If $P_1 \times \ldots \times P_n$ is denoted as Prods and $\tilde{P}_1 \sqcap \ldots \sqcap \tilde{P}_n$ is denoted as Inters, the lemma 4.5 could be proved as follows:

We shall show that for any state $(s_{i_1}^{P_1}, s_{i_2}^{P_2}, \ldots, s_{i_n}^{P_n})$ of Prods and there exists a state $(\tilde{s}_{i_1}^{P_1}, \tilde{s}_{i_2}^{P_2}, \ldots, \tilde{s}_{i_n}^{P_n})$ of Inters such that $(s_{i_1}^{P_1}, s_{i_2}^{P_2}, \ldots, s_{i_n}^{P_n}) \lesssim\gtrsim (\tilde{s}_{i_1}^{P_1}, \tilde{s}_{i_2}^{P_2}, \ldots, \tilde{s}_{i_n}^{P_n})$.

($\Rightarrow$) For a transition relation from the state of Prods:
$\{((s_{i_1}^{P_1}, \ldots, s_{i_i}^{P_i}, \ldots, s_{i_n}^{P_n}), a, (s_{i_1}^{P_1}, \ldots, s_{i_{i+1}}^{P_i}, \ldots, s_{i_n}^{P_n})) | (s_{i_i}, a, s_{i_{i+1}}) \in \Delta_{P_i}\}$.
From the construction of Inters, we could find that the corresponding automata of P_i is $\tilde{P}_i$ such that $(\tilde{s}_{i_i}, a, \tilde{s}_{i_{i+1}}) \in \tilde{\Delta}_{P_i}$
Because of the pairwise disjoint alphabet, for other automaton $P_j, j \neq i$, $(\tilde{s}_i, a, \tilde{s}_{i_{i+1}}) \notin \tilde{\Delta}_{P_j}$ and hence $(\tilde{s}_{i_i}, a, \tilde{s}_{i_i}) \in \tilde{\Delta}_{P_j}$
As a result, there is always a corresponding transition relation
$\{((\tilde{s}_{i_1}^{P_1}, \ldots, \tilde{s}_{i_i}^{P_i}, \ldots, \tilde{s}_{i_n}^{P_n}), a, (\tilde{s}_{i_1}^{P_1}, \ldots, \tilde{s}_{i_{i+1}}^{P_i}, \ldots, \tilde{s}_{i_n}^{P_n})) | (\tilde{s}_{i_i}, a, \tilde{s}_{i_{i+1}}) \in \tilde{\Delta}_{P_i}\}$.
By induction, we could get that Prods $\lesssim$ Inters if recursively considering all the outgoing transitions from each pair of states.

($\Leftarrow$) Similar to ($\Rightarrow$).

□

As a direct consequence of this lemma, the problem of checking whether $\tilde{P}_1 \sqcap \ldots \sqcap \tilde{P}_n \lesssim P$ is EXPTIME-hard. We reduce this latter problem to a problem of checking simulation between two pd-processes P_A and P_B. The construction of P_A and P_B is described below.

Construction of P_A and P_B.

For each automaton $\tilde{P}_i = (\tilde{S}_{P_i}, \tilde{s}_0^{P_i}, \tilde{Act}_{P_i}, \tilde{\Delta}_{P_i}, \tilde{F}_{P_i})$ we construct a pd-process $\hat{P}_i = (\hat{S}_{P_{A_i}}, \hat{s}_0^{P_i}, \hat{Act}_{P_i}, D, \hat{G}_{P_i}, \hat{\Delta}_{P_i}, \hat{F}_{P_i})$ such that:

- $\hat{S}_{P_i} = \tilde{S}_{P_i}$,$\hat{s}_0^{P_i} = \tilde{s}_0^{P_i}$, $\hat{F}_{P_i} = \tilde{F}_{P_i}$ and $\hat{Act}_{P_i} = \tilde{Act}_{P_i}$;
- D is the probabilistic database depicted at Table 4.1. It is defined over a schema made of a unary relation Test which contains the tuples $t_{g_A}, t_{g_B}, t_1, \ldots t_n$. The probabilities of these tuples are computed as follows:
  - $\forall i \in [1, n]$, $x_i = v$, for any value v s.t. $\frac{2}{2n-1} < v < 1$, (w.l.o.g., we assume $n \geq 2$).
  - x_{g_A} takes any value satisfying $0 < x_{g_A} < \frac{2}{(2n-1)v}$.
  - $x_{g_B} = (n - \frac{1}{2}) * v * x_{g_A}$.
- $\hat{G}_{P_i} = \{g_A, g_i\}$ is the set of guards. The guards g_A and g_i are defined as follows:
  $\mathsf{g_A}() : -\mathsf{Test}(\text{`}\mathsf{g'_A})$
  $\mathsf{g_i}()$:- $\neg\mathsf{Test}(\text{`}\mathsf{g'_1}), \ldots, \neg\mathsf{Test}(\text{`}\mathsf{g'_{i-1}}), \mathsf{Test}(\text{`}\mathsf{g'_i}), \neg\mathsf{Test}(\text{`}\mathsf{g'_{i+1}}), \ldots, \neg\mathsf{Test}(\text{`}\mathsf{g'_n})$
- $\hat{\Delta}_{P_i} = \{(s_j, a, g_i, s_j) \mid \exists (s_j, a, s_j) \in \Delta_{P_i}$ and $a \in \tilde{Act}_{P_i}\} \cup \{(s_j, a, g_A, s_j) \mid \exists (s_j, a, s_j) \in \Delta_{P_i}$ and $a \in \bigcup_{\substack{j=1 \\ j \neq i}}^{n} \tilde{Act}_{P_j}\}$

Construction of P_A. We merge together the $\hat{P}_i$s pd-processes to form the pd-process $P_A = (S_{P_A}, s_0^{P_A}, Act_{P_A}, D, G_{P_A}, \Delta_{P_A}, F_{P_a})$ defined as follows:

- $s_0^{P_A}$ is a new state used as the initial state of P_A.
- $S_{P_A} = \{s_0^{P_A}\} \cup \bigcup_{i=1}^{n} \hat{S}_{P_{A_i}}$.
- $Act_{P_A} = \{a_{new}\} \cup \bigcup_{i=1}^{n} \hat{Act}_{P_i}$, where a_{new} is a new action.
- $G_{P_A} = \bigcup_{i=1}^{n} \hat{G}_{P_i}$.

Relation Test

	Attrib	Pr
t_{g_A}	g_A	x_{g_A}
t_{g_B}	g_B	x_{g_B}
t_1	g_1	x_1
t_2	g_2	x_2
	...	...
t_n	g_n	x_n

TABLE 4.1: The probabilistic database D

- $\Delta_{P_A} = \{(s_0^{P_A}, a_{new}, true, \hat{s}_0^{P_i}),\ \text{for } i \in [1,n]\} \cup \bigcup_{i=1}^{n} \hat{\Delta}_{P_i}$.
- $F_{P_i} = \bigcup_{i=1}^{n} \hat{F}_{P_i}$.

Construction of P_B. Given an automaton $P = (S_P, s_0^P, Act_P, \Delta_P, F_P)$, we construct a pd-process $P_B = (S_{P_B}, s_0^{P_B}, Act_{P_B}, D, G_{P_B}, \Delta_{P_B}, F_{P_B})$ as follows.

- $s_0^{P_B}$ is a new state used as the initial state of P_B.
- $S_{P_B} = \{s_0^{P_B}\} \cup S_P$.
- $Act_{P_B} = \{a_{new}\} \cup Act_P \cup \bigcup_{i=1}^{n} Act_{P_i}$.
- $G_{P_B} = \{g_b\}$
- $\Delta_{P_B} = \{(s_0^{P_B}, a_{new}, true, s_0^P)\} \cup \{(s, a, true, s') \mid (s, a, s') \in \Delta_P\} \cup \{(s_0^{P_B}, a, g_b, s_0^{P_B}) \mid a \in \{a_{new}\} \cup \bigcup_{i=1}^{n} Act_{P_i}\}$
- $F_{P_B} = F_P \cup \{s_0^{P_B}\}$.
- D is the same probabilistic database used for P_A.

Observe that, by construction, $|P_A|$ is linear in the size of $P_i s$ while $|P_B|$ is linear in the sizes of the $P_i s$ and P.

□

Lemma 4.6. *Let P, P_i, with $i \in [1,n]$, a set of automata, the $P_i s$ having disjoint alphabets. Let P_A and P_B the corresponding pd-processes constructed as described above. Then: $\tilde{P}_1 \sqcap \ldots \sqcap \tilde{P}_n \lesssim P$ iff $P_A \lesssim P_B$*

Proof. of lemma 4.6. We make first the following observations.

- Since $G_{P_B} = \{g_b\}$, the world-partition automata of P_B contains only the following two automata:
  - $\hat{P}$ representing the partition associated with the guard $\neg g_b$. $\tilde{P}$ corresponds to the automata P augmented with a new initial state $s_0^{P_B}$ and a transition labelled a_{new} from $s_0^{P_B}$ to the initial state of P.
  - an automaton, noted P^u, representing the partition associated with the guard g_b. Note that the automata P^u is universal since it contains in its initial state $s_0^{P_B}$, which is also a final state, a loop labelled with alphabet from $\{a_{new}\} \cup \bigcup_{i=1}^{n} Act_{P_i}$. Hence, P^u simulates any automaton from the world-partition automata of P_A.
- Note that the guards g_i in P_A are pairwise disjoint. Hence, the world-partition automata of P_A contains two subsets:
  - A set, noted Pos_A, of n automata representing partitions where g_a appears positively.
  - A set, noted Neg_A, of n automata representing partitions in which the guard g_a appears negatively.

Let us now prove the two directions of lemma 4.6.

($\Rightarrow$) Assume that $\tilde{P}_1 \sqcap \ldots \sqcap \tilde{P}_n \lesssim P$

$\Rightarrow \hat{P}_1 \sqcap \ldots \sqcap \hat{P}_n \lesssim \hat{P}$ (by construction of $\hat{P}$ and the $\hat{P}_i$s),

$\Rightarrow \forall T \in Neg_A,\ T \lesssim \hat{P}$,

$\Rightarrow \forall T \in Neg_A,\ T \sqcap \prod_{\substack{\hat{P}_j \in X \\ X \subseteq Pos_A \cup Neg_A}} \hat{P}_j \lesssim \hat{P}$,

Note that $\max_{T \in C_L(P_{A_{P_G}})} (Prob(T)) = Prob(g_A) + Prob(\neg g_A) = 1$

$\Rightarrow \max_{T \in C_L(P_{A_{P_G}})} (Prob(T)) \leq Prob(\hat{P}) + Prob(P^u)$

$\Rightarrow P_A \lesssim P_B$

($\Leftarrow$) Assume that $P_A \lesssim P_B$

$\Rightarrow \hat{P}_1 \sqcap \ldots \sqcap \hat{P}_n \lesssim \hat{P}$. This is because P^u alone is not enough to simulate $\hat{P}_1 \sqcap \ldots \sqcap \hat{P}_n$ since $Pr(P^u) = x_{g_B} = (n - \frac{1}{2}) * v * x_{g_A} \leq n * v * x_{g_A} = Pr(\hat{P}_1 \sqcap \ldots \sqcap \hat{P}_n)$.

$\Rightarrow \tilde{P}_1 \sqcap \ldots \sqcap \tilde{P}_n \lesssim P$ $\square$

4.2 Model checking

This section proposes the method to do model checking in the context of pd-processes. Because pd-process is a probabilistic transition system, LTL or P-LTL captures the parallel meaning as well as CTL or P-CTL. In this thesis, we only consider P-LTL and P-CTL model checking. The first part is about the discussion of algorithm and complexity of the P-LTL model checking over pd-processes. The second part is denoted to P-CTL algorithms and complexity.

To make the description clear, this chapter uses the same example as the previous chapters. But model checking focuses on the proposition of every state. Figure 4.1 illustrates the atomic propositions and their distribution for every state.

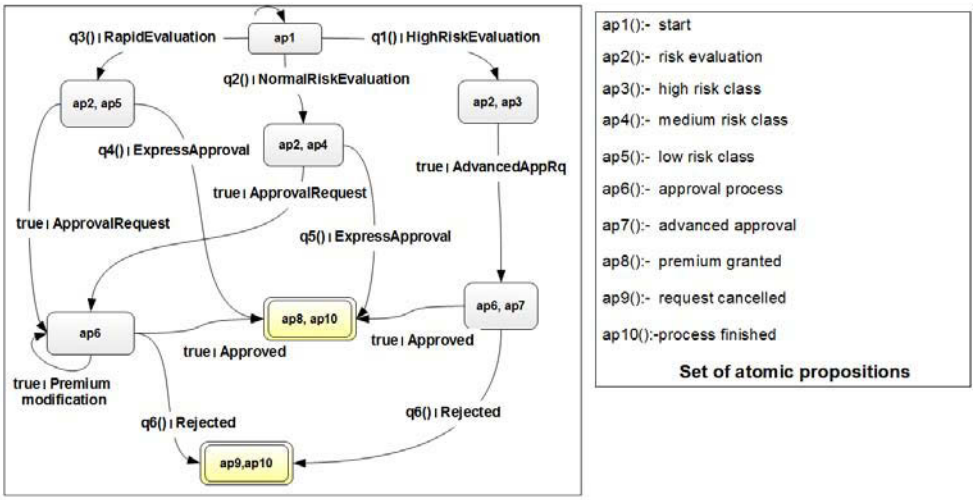

FIGURE 4.1: The proposition assignment for every state.

4.2.1 P-LTL Model Checking on Pd-processes

Because pd-process is a probabilistic transition system, linear temporal properties are described in the fashion of checking probability and LTL formulae. As a result, linear temporal properties are always related to probabilistic model checking. In this thesis, we don't distinguish LTL or P-LTL model checking in the context of pd-processes. So checking linear temporal properties over pd-processes is named as P-LTL model checking on pd-processes. By reusing the syntax of P-LTL and following the traditional P-LTL model checking method, the method of P-LTL model checking on pd-processes is as follows. Let $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ be a pd-process and let $W \in \mathcal{W}(D)$, we denote $A_w = (S, s_0, D_W, Act, G, \Delta, F, AP, L)$ be an automaton associated with world W. D_W can be seen as a probabilistic database derived from W where the probability

of each tuple in W is 1. The definition of P-LTL model checking on Pd-processes is as follows:

Definition 4.7 (P-LTL Model Checking on Pd-processes)**.** For a given pd-process $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ and a P-LTL formula φ, $A \vDash_{\sim pr} \varphi$ iff $\sum_{\substack{A_{W_i} \vDash \varphi \\ W_i \in \mathcal{W}(D)}} Pr(A_{W_i}) \sim pr$ ($\sim$ means $=, <, >, \leqslant, \geqslant$).

Because the paths of pd-processes are probabilistic, the P-LTL model checking on pd-processes cannot directly check the paths of original pd-process. Fortunately, a pd-process is a set of traditional automata corresponding with the possible worlds of probabilistic database and these automata are adequate through P-LTL model checking method. As a result, the P-LTL model checking on pd-processes is slightly changed to check if it is possible to return a "true" answer satisfying the given probability. Following the example in Figure 4.1 and probabilistic database in Table 2.2, we want to check some properties in this model. First, these properties are described in nature language as follows:

- Checking if the premium application can be approved and give answers at the end of approval processes with probability > 0.5.
- Checking if the high risk evaluation can be approved with probability > 0.2.

We interpret these properties to LTL formula. For $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$, the properties above can be translated to

1. $Pr(A \vDash \bigcirc ap_6 \wedge \Diamond ap_{10}) > 0.5$.
2. $Pr(A \vDash ap_3 \wedge \Diamond ap_8) > 0.2$.

Algorithm 2 depicts the method of P-LTL checking under the context of pd-process which stands for a finite set of traditional automata associated with different probabilities represented by a set of partition automata. To do the P-LTL model checking under this context, we could do it for every partition automaton by following the traditional LTL model checking methodology. Because of the existence of probability for each partition automaton, the P-LTL model checking may not be 100% true or false. To accumulate the probability of "true", we introduce a variable P to help us checking the final probability satisfaction.

Proof. For a given pd-process $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$, if $W(P_i)$ stands for the possible worlds which could be expressed by guard partition P_i, $W(P_i) = \{W_{i_1}, W_{i_2}$

Algorithm 2 Algorithm of P-LTL model checking on pd-processes

Require:

A pd-process $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ and P-LTL formula φ over AP, checking if $A \vDash_{\sim pr} \varphi$.

Ensure:

1: Construct a Non-deterministic Buchi Automata $NBA_{\neg\varphi}$ such that $L(NBA_{\neg\varphi}) = Words(\neg\varphi)$ and a variable P to record the probabilities of adequate partition automata.
2: Generate partition automata A_{P_G} for A and for each partition automaton $A_{P_i} \in A_{P_G}$, construct the product $A_{P_i} \otimes NBA_{\neg\varphi}$.
3: If there dose not exist any accepted sequence (for example, a path Π) in $A_{P_i} \otimes NBA_{\neg\varphi}$, it means the model checking will return "true" with respect to the partition automata A_{P_i}, $P+ = Pr(A_{P_i})$.
4: If $P \sim pr$, return true, otherwise return false.
5: **return** True or false .

$, ..., W_{i_j}\}$ where $W_{i_k} \in \mathcal{W}(D), 1 \leqslant k \leqslant j$. Then we have $\{A_{W_{i_1}}, A_{W_{i_2}}, ..., A_{W_{i_j}}\}$ corresponding with the worlds in $W(P_i)$. Because for any $W_{i_k} \in W(P_i)$, it can make P_i true. A_{P_i} and $A_{W_{i_k}}$ are equivalent automata and by definition of world partition automata, we have $Pr(A_{P_i}) = \sum_{k=1}^{j} Pr(A_{W_{i_k}})$. Meanwhile, by definition of guard partitions, if $W_{i_k} \in W(P_i)$, then $W_{i_k} \notin W(P_j)$ where $P_i \neq P_j$. Obviously, for a given P-LTL formula φ, if $\exists A_{W_{i_k}} \vDash \varphi$, $W_{i_k} \in W(P_i)$, then $\forall A_{W_{i_m}} \vDash \varphi, W_{i_m} \in W(P_i)$. If $\sum_{\substack{A_{W_i} \vDash \varphi \\ W_i \in \mathcal{W}(D)}} Pr(A_{W_i}) \sim pr$, there exists a set of world partition automata $\{A_{P_{i_1}}, A_{P_{i_2}}, ..., A_{P_{i_j}}\}$ such that $W_i \in W(P_{i_k}), 1 \leqslant k \leqslant j$ and $\sum_{A_{P_i} \vDash \varphi} Pr(A_{P_i}) \sim pr$. Therefore, Algorithm 2 and Definition 4.7 are equivalent. □

Continuing checking the formulae of the example above, do LTL model checking for every partition automaton to check the satisfaction of these two P-LTL formulae. Taking the first one to briefly be studied, taking the partition automata in Fig. 3.4 (c) as an example, both automata can be satisfied by formula 1. Then we could add the probability, it is easy to see the final probability is bigger than 0.5. So formula 1 is satisfied. Then we consider formula 2, in Fig. 3.4 (c), both automata are not satisfied by formula 2. When we check all the partition automata, only the one with guard partitions $\{g_1\}$ can be satisfied, the probability is 8.68%. As a result, formula 2 is not satisfied due to the probability is less than what we need.

Complexity of P-LTL model checking on pd-processes

Theorem 4.8. *The complexity of P-LTL model checking on pd-processes is as follows:*
(i) in $\mathcal{O}(f(|D|))$ in data complexity, where $f(|D|)$ is the data-complexity of computing the probabilities of a boolean query on a probabilistic database D,
(ii) EXPTIME*-complete w.r.t. the expression complexity,*

Proof. **Proof of (ii)** From [17], we already know that checking a P-LTL formula can be achieved in EXPTIME w.r.t expression complexity. According to theorem 4.8, do P-LTL model checking on pd-process A of P-LTL formula φ, can be achieved in several steps:

- Computing world-partition automata of A_{P_G}. The size of A_{P_G} is bounded by $2^{|G|}$ and, from definition 3.5, one can derive that each automata in A_{P_G} has a size bounded by $|A|$ and can be constructed: (i) in time linear in $|A|$, and (ii) by evaluating one probabilistic query q.
- For every world-partition automaton A_{P_i}, do the traditional LTL model checking. The complexity of traditional LTL model checking is bounded by $|A| * 2^{|\varphi|}$.
- To sum up, the P-LTL model checking equals that doing traditional LTL model checking on every partition automata. Hence, the complexity is $O(|A| \times 2^{|\varphi|} \times 2^{|G|})$.

The hardness is obtained from the verification of standard LTL model checking which is known as EXPTIME-complete. Indeed, such a problem can be easily translated by letting a pd-process with an empty probabilistic database and setting all the guards to true.

Proof of (i) From proof of (ii), the number of query to be evaluated over the probabilistic database is bounded by $2^{|G|}$ (computation of A_{P_G}). Hence, assuming $|A|$ to be a constant enables to check LTL properties with a data-complexity that matches the complexity of probabilistic query evaluation. □

4.2.2 P-CTL Model Checking on Pd-processes

At present, the problem is how to achieve P-CTL model checking under the context of pd-process. If we consider the P-CTL model checking method is a black box function and assume a pd-process is a set of traditional automata (partition automata) without guards, the result of model checking for each partition automaton is a set of states which satisfy the P-CTL formulae. The semantic of traditional P-CTL model checking is to verify the set of states which satisfy the CTL formulae and also the probability given by the syntax $P_{\sim pr}(\phi)$. This probability restricts the probability scope of the candidate states. So under the context of pd-processes, we can reuse the method of CTL model checking of pd-processes. That's because the semantic of pd-processes is for a given probabilistic database, different possible worlds can let corresponding pd-process represents different automata (partition automata). The probability of each automaton means the probability of the occurrence of the states in this automaton. If one state exists in several partition automata, that means the occurrence of this states

in the overall pd-process is the sum of the corresponding automata probabilities. When we go back to the problem of P-CTL model checking, the syntax $P_{\sim pr}(\phi)$ restricts the probability of the states satisfying ϕ. Under the context of pd-process, if s_i satisfies ϕ in a set of partition automata, the probability of s_i satisfying ϕ is undoubtedly the sum of these corresponding partition automata. The definition of P-CTL model checking on pd-processes is as follows:

Definition 4.9 (PCTL Model Checking on Pd-processes)**.** For a given pd-process $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ and a P-CTL formula $P_{\sim pr}(\Phi)$, the satisfaction set is $Sat(P_{\sim pr}(\Phi) = \{s | s \in S, s \vDash \Phi, Pr(s) \sim pr\}$where $Pr(s) = \sum Pr(A_{W_i}) | W_i \in \mathcal{W}(D), s \in S_{A_{W_i}}$ and $s \vDash \Phi$ under the context of A_{W_i}. ($\sim$ means $=, <, >, \leqslant, \geqslant$).

Because $P_{\sim pr}(\phi)$ is a P-CTL syntax and it can be a part of Φ, under the context of pd-processes, it is impossible to reuse the traditional P-CTL model checking algorithm, we could reuse the method of traditional CTL model: checking to build the parse tree of P-CTL formula and computing the satisfaction set of each node. To compute the probability and merge the result of child node in the parse tree, there needs a set to restore the adequate partition automata whose set of states satisfy the P-CTL syntax. The algorithms for non-probabilistic syntax and probabilistic syntax is as follows:

Algorithm 3 Algorithm of computing satisfaction set of P-CTL syntax:probabilistic case

Require:

A pd-process $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ and P-CTL formula $P_{\sim pr}(\phi)$ over AP, finding $Sat(P_{\sim pr}(\phi))$.

Ensure:

1: Generate a set to record the partition automata whose states satisfy the syntax denoted as $SatPA(P_{\sim pr}(\phi))$.
2: Do traditional CTL model checking on every partition automaton A_{P_i} and if $s_i \in S_{A_{P_i}}$, $s_i \vDash \phi$ under the context of A_{P_i}, $Pr(s_i) += Pr(A_{P_i})$, $SatPA(P_{\sim pr}(\phi))$ adds A_{P_i}.
3: Finally, checking all the states, if $Pr(s_i) \sim pr$, $Sat(P_{\sim pr}(\phi))$ adds s_i.
4: **return** $Sat(P_{\sim pr}(\phi))$ and $SatPA(P_{\sim pr}(\phi))$.

With the help of Algorithm 3 and Algorithm 4, the algorithm to do the P-CTL model checking can be started from dividing the P-CTL formula in the form of parse tree. Then we can bottom-up compute the satisfaction set for each node. For the leaf nodes, using Algorithm 3 and Algorithm 4 compute the satisfaction set of states and the satisfaction set of partition automata. For the internal node, if non-probabilistic case, intersect these two sets from its child nodes; else if probabilistic case, intersect these two sets then check the probability of each state in the satisfaction set of states according to its occurrence in the satisfaction set of partition automata. Finally, the satisfaction of root is the answer of model checking. The method is described in Algorithm 5.

Algorithm 4 Algorithm of computing satisfaction set of P-CTL syntax:non-probabilistic case

Require:

A pd-process $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ and CTL formula ϕ over AP, finding $Sat(\phi)$.

Ensure:

1: Generate a set to record the partition automata whose states satisfy the syntax denoted as $SatPA(\phi)$.
2: Do traditional CTL model checking on every partition automaton A_{P_i} and if $s_i \in S_{A_{P_i}}$, $s_i \vDash \phi$ under the context of A_{P_i}, $Pr(s_i) + = Pr(A_{P_i})$, $Sat(\phi))$ adds s_i, $SatPA(\phi)$ adds A_{P_i}.
3: **return** $Sat(\phi))$ and $SatPA(\phi)$.

Algorithm 5 Algorithm of P-CTL model checking on pd-processes

Require:

A pd-process $A = (S, s_0, D, Act, G, \Delta, F, AP, L)$ and P-CTL formula $P_{\sim pr}(\Phi)$ over AP.

Ensure:

1: Build the parse tree of P-CTL formula.
2: Bottom up compute the satisfaction set for each node.
3: For leaf nodes, if the syntax ϕ is like $P_{\sim pr}(\phi)$, use Algorithm 3, otherwise, use Algorithm 4, computing $SatPA(\phi)$ and $Sat(\phi)$.
4: For internal nodes, if the syntax Φ_i is a non-probabilistic syntax, intersect the $Sat(\phi)$ for each child nodes to generate $Sat(\Phi_i)$ and do the same to generate $SatPA(\Phi_i)$, otherwise, not only generating $Sat(\Phi_i)$ and $SatPA(\Phi_i)$ but also checking every state $s_i \in Sat(\Phi_i)$, if $\sum_{A_{P_i} \in SatPA(\Phi_i)} Pr(A_{P_i}) | s_i \in A_{P_i}$ dose not satisfy the need of Φ_i, $Sat(\Phi_i)/s_i$
5: The satisfaction set of root is the answer of P-CTL model checking.
6: **return** $Sat(P_{\sim pr}(\Phi))$.

The proof is similar with the one of P-LTL model checking on pd-processes. For a given pd-process A and a P-CTL formula Φ, we check every state s_i whether $s_i \vDash \Phi$ and $Pr(s_i) \sim pr$. By Definition 4.9, we check every automaton $A_{W_i} = (S, s_0, D_{W_i}, Act, G, \Delta, F, AP, L)$ corresponding with a given world $W_i \in \mathcal{W}(D)$, where $s_i \in S$ of A_{W_i}. From the proof of Algorithm 2, we find that A_{W_i} can be represented as its associated world partition automaton. Then we can transform this problem to check all the world partition automata. It is obviously that Algorithm 5 and Definition 4.9 are equivalent.

4.2.2.1 Complexity of P-CTL model checking on pd-processes

Theorem 4.10. *The complexity of P-CTL model checking on pd-processes is as follows:*
(i) in $\mathcal{O}(f(|D|))$ *in data complexity, where* $f(|D|)$ *is the data-complexity of computing the probabilities of a boolean query on a probabilistic database* D*,*
(ii) EXPTIME *w.r.t. the expression complexity,*

(iii) can be solved in EXPTIME *w.r.t.* $O((|A|) \times |\Phi|^2 \times 2^{|G|})$ *which is capable to be abbreviated by* $O(2^{|G|})$.

Proof. **Proof of (iii)** According to theorem 4.10, do P-CTL model checking on pd-process A of P-CTL formula Φ, can be achieved in several steps:

- Computing world-partition automata of A_{P_G}. The size of A_{P_G} is bounded by $2^{|G|}$ and, from definition 3.5, one can derive that each automata in A_{P_G} has a size bounded by $|A|$ and can be constructed: (i) in time linear in $|A|$, and (ii) by evaluating one probabilistic query q.
- For every world-partition automaton A_{P_i}, do the traditional CTL model checking. The complexity of traditional CTL model checking is bounded by $|A| \times |\Phi|^2$.
- To sum up, the P-CTL model checking equals that doing traditional CTL model checking on every partition automata. Hence, the complexity is $O((|A|) \times |\Phi|^2 \times 2^{|G|})$. If we abbreviate it, the final complexity is $O(2^{|G|})$.

Proof of (i) From proof of (iii), the number of query to be evaluated over the probabilistic database is bounded by $2^{|G|}$ (computation of A_{P_G}). Hence, assuming $|A|$ to be a constant enables to check LTL properties with a data-complexity that matches the complexity of probabilistic query evaluation.
Proof of (ii) Because the complexity of computing CTL model checking is in PTIME but we need to do CTL model checking in every partition automaton, computing partition automata is in EXPTIME. As a result, the total expression complexity is EXPTIME. □

4.2.2.2 Example

We continue using the example of premium calculation. The property we want to check is if there exists any state being a risk evaluation letting next state is expert evaluation and finally this premium calculation is permitted with probability > 0.3. Translate it to P-CTL formula: $\exists(((ap_2 \wedge \bigcirc ap_7)) \wedge (Pr(\Diamond ap_8) > 0.3))$. Then we build parse tree for this P-CTL formula in Figure 4.2. Bottom-up finding the satisfaction set for every partition automaton, we take the partition automata in Fig. 3.4 (c) as an example again. First considering ap_2, the satisfaction set for Ap_1 is $\{s_2\}$, for Ap_2 is $\{s_2, s_3\}$, within this two partition automata, $Sat(ap_2) = \{s_2, s_3\}$, $SatAP(ap_2) = \{Ap_1, Ap_2\}$, then we check all the other partition automata, $Sat(ap_2) = \{s_1, s_2, s_3\}$. Following the same manner, $Sat(\bigcirc ap_7) = \{s_3\}$, then we merge the result of these two leaves in the

parse tree. $Sat(ap_2 \wedge \bigcirc ap_7) = \{s_2\}$. Noticing that, the set of satisfaction partition automata is updated as well. Because there are too many result, we don't show it here. When we go to find the satisfaction set of the leaf $Sat(Pr(\Diamond ap_8) > 0.3)$, the probability of every state satisfying $\Diamond ap_8$ is considered with respect to the satisfaction partition automata of $Sat(Pr(\Diamond ap_8) > 0.3)$. The answer is that the satisfaction set of $Sat(Pr(\Diamond ap_8) > 0.3)$ is $\{s_0, ..., s_5\}$ regardless probability, then considering probability, the final answer is $\{s_0, s_1, s_2, s_3\}$. Finally, we merge the two satisfaction set $Sat(ap_2 \wedge \bigcirc ap_7)$ and $Sat(Pr(\Diamond ap_8) > 0.3)$, the result is $\{s_2\}$. As a result, the satisfaction set of $Sat(\exists(((ap_2 \wedge \bigcirc ap_7)) \wedge (Pr(\Diamond ap_8) > 0.3))) = \{s_2\}$.

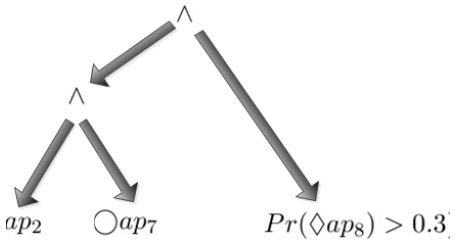

FIGURE 4.2: The parse tree of P-CTL formula $\exists(((ap_2 \wedge \bigcirc ap_7)) \wedge (Pr(\Diamond ap_8) > 0.3))$.

Chapter 5

Optimized Algorithms

In this chapter, several optimized algorithms of testing simulation relation on pd-processes are studied under different contexts. Deterministic pd-processes, pseudo-deterministic pd-processes and strong non-deterministic pd-processes classify pd-processes to several different but related scopes. The third section introduces a special case: the simulation relation test on a compiled approach when only the content of query or database is altered. Last section reveals the connection between pd-processes with Markov processes. As the previous chapters described, pd-process is a non-Markov process but some special pd-processes which satisfy some properties are equivalent to Markov processes. We present the notion of independent pd-processes which can be considered as Markov processes. In the context of independent pd-processes, we have a lower complexity to do the simulation relation test and model checking.

5.1 Deterministic pd-processes

A formal definition of deterministic pd-processes is given in Definition 5.1.

Definition 5.1 (Deterministic pd-process)**.** Let $A = \{S, s_0, D, Act, \Delta, G, F, AP, L\}$ be a pd-process, A is deterministic iff $\forall s_i \in S$ and $\forall(s_i, a_i, g_i, s_{i_1}) \in \Delta, \forall(s_i, a_j, g_j, s_{i_2}) \in \Delta$ and $s_{i_1} \neq s_{i_2}$then $a_i \neq a_j$.

Under the context of Definition 5.1, we build the partition automata following Theorem 4.3 according to the partitions of guards. The partitions of guards are the power set of all the guards. To ensure the partition automata cover all the probability, the closure automata which are the intersections of the partition automata are considered. Lemma 5.3 states the fact that the intersection among partition automata from a deterministic

pd-process do not create new automaton which differs from the partition automata involved into this intersection.

Proposition 5.2. *For two partition automata* $\{A_{P_i}, A_{P_j}\}$ *of a deterministic pd-process* A, $\forall A_{P_i}, A_{P_j} \in A_{P_G}$, $\exists A_{P_k}$ *with* $P_k = P_i \cap P_j$ *such that* $Tr(A_{P_i} \sqcap A_{P_j}) = Tr(A_{P_k})$.

By expressing in terms of execution trees, Proposition 5.2 states a fact that if the result of intersection between two partition automata of a deterministic pd-process is not empty, then there exists another partition automaton which coincides with this result. The proof of this proposition is given below.

Proof. To prove this proposition, let us propose a property of deterministic automata first: because the actions of transition relations outgoing from one state are always different, for the world $W(P_k)$ of a given partition of guard P_k, there exist a set of execution trees $Tr(A_{P_k})$ associated with only one partition automaton A_{P_k}. If a world $W(P_i) \supset W(P_k)$, $W(P_i)$ associating with a partition automaton A_{P_i}, $\forall T_k \in Tr(A_{P_k})$, $\exists T_i \in Tr(A_{P_i})$ such that T_k is a sub-tree of T_i.
Here is the proof for this property. Because for any path $s_{i_1} \xrightarrow{a_{i_1}, g_{i_1}} s_{i_2} \xrightarrow{a_{i_2}, g_{i_2}} s_{i_3} \ldots \xrightarrow{a_{i_j}, g_{i_j}} s_{i_{j+1}}$ in a deterministic pd-process A, we cannot find another path through different states assigned with the same sequence of actions and guards. That is to say for two world partition automata A_{P_k} and A_{P_i} of A, if $P_k \subset P_i$, $\forall \pi_k \in A_{P_k}$ then $\pi_k \in A_{P_i}$.
After this property, we prove this proposition as follows:

- By Definition 5.1, it is obviously that $\forall A_{P_i}, A_{P_j} \in A_{P_G}$, $P_i \cap P_j \neq \emptyset$, $\exists A_{P_k}$ with $P_k = P_i \cap P_j$.
- $\Leftarrow$, according to the property above, if $A_{P_i} \sqcap A_{P_j} \neq \emptyset$, $P_k = P_i \cap P_j$ then $Tr(A_{P_k}) \subset Tr(A_{P_i})$ and $Tr(A_{P_k}) \subset Tr(A_{P_j})$. As a result, we have $Tr(A_{P_i} \sqcap A_{P_j}) \supseteq Tr(A_{P_k})$.
- $\Rightarrow$, similar with the proof of the property above, if $A_{P_i} \sqcap A_{P_j} \neq \emptyset$, $P_k = P_i \cap P_j$ then for any path $s_{i_1} \xrightarrow{a_{i_1}, g_{i_1}} s_{i_2} \xrightarrow{a_{i_2}, g_{i_2}} s_{i_3} \ldots \xrightarrow{a_{i_j}, g_{i_j}} s_{i_{j+1}}$ in A, if its corresponding path $\pi_i = s_{i_1} \xrightarrow{a_{i_1}} s_{i_2} \xrightarrow{a_{i_2}} s_{i_3} \ldots \xrightarrow{a_{i_j}} s_{i_{j+1}} \in A_{P_i} \sqcap A_{P_j}$, then $\pi_i \in A_{P_k}$. Then we have: for any tree T_i if $T_i \in Tr(A_{P_i})$ and $T_i \in Tr(A_{P_j})$, $T_i \in Tr(A_{P_k})$. As a result, $Tr(A_{P_i} \sqcap A_{P_j}) \subseteq Tr(A_{P_k})$.

Finally ,we have $Tr(A_{P_i} \sqcap A_{P_j}) = Tr(A_{P_k})$.

□

Lemma 5.3. *Let* A *be a deterministic pd-process,* $C_L(A) \setminus A_{P_G} = \emptyset$.

Proof. According to Proposition 5.2, for a set of partition automata $\{A_{P_{i_1}}, A_{P_{i_2}}, ..., A_{P_{i_n}}\}$ of a deterministic pd-process A, if $\bigsqcap_{k=1}^{n}(A_{P_{i_k}}) \neq \emptyset$, there exists $A_{P_j} \in A_{P_G}$ such that $Tr(A_{P_j}) = \bigsqcap_{k=1}^{n}(Tr(A_{P_{G_{i_k}}}))$ and $P_j \subseteq \bigcap_{k=1}^{n}(P_{i_k})$ That is to say: all the closure automata of A can be represented by partition automata A_{P_G}. □

Following Lemma 5.3, under the context of deterministic pd-process, all the closure automata are contained with partition automata so there is no need to compute closure automata. However, in order to include the probabilities of related closure automata, the probabilities of the partition automata need to be recomputed. The computation is achieved as follows. For each partition automaton A_{P_i}, it can be seen as a result of intersecting all the partition automata whose execution trees contain $Tr(A_{P_i})$. That is to say the probability of A_{P_i} is the sum of the probabilities of all these partition automata. Because this way of probability computation only consider the positive part of guards, to distinguish with $Pr(A_{P_i})$ defined in the previous chapters, we denote this kind of probabilities as $Pr(A_{P_i})^{pos}$. $Pr(A_{P_i})^{pos} = \prod_{q_j \in P_i} Pr(q_j)$. Lemma 5.4 states the reason why we could compute the probability in this way.

Lemma 5.4. *Let A be a deterministic pd-process, $T \in Tr(A)$ with $Pr(T, A) > 0$ iff $\exists A_{P_i} \in A_{P_G} | T \in Tr(A_{P_i})$ and $Pr(T, A) = Pr(A_{P_i})^{pos}$*

Proof. For a given pd-processes $A = \{S, s_0, D, Act, G, \Delta, F\}$, $G = \{q_1, q_2, ..., q_n\}$, P_G is the set of guard partitions.
According to Lemma 3.6, $T \in Tr(A)$ with $Pr(T, A) > 0$ iff $\exists \{A_{P_j}, \ldots, A_{P_n}\} \subseteq A_{P_G}$ such that: $T \in Tr(A_{P_k})$, $\forall k \in \{j, \ldots, n\}$, and $Pr(T, A) = \sum_{k \in \{j,...,n\}} Pr(A_{P_k})$. Assuming P_i accords to the condition of Lemma 5.4 and $P_i = \{q_1, q_2, ..., q_i\}$.
$Pr(P_i) = \prod_{j=1}^{i} Pr(q_j) \times \prod_{k=i+1}^{n}(1 - Pr(q_k))$.
Enumerate all the guard partitions which are the super set of P_i in P_G such as $P_{i+1} = \{q_1, q_2, ..., q_i, q_{i+1}\}$, $P_n = \{q_1, q_2, ..., q_i, ..., q_n\}$, etc.
$\sum_{k \in \{j,...,n\}} Pr(A_{P_k}) =$
$\prod_{j=1}^{i} Pr(q_j) \times \prod_{k=i+1}^{n}(1 - Pr(q_k))+$
$\prod_{j=1}^{i} Pr(q_j) \times Pr(q_{i+1}) \times \prod_{k=i+2}^{n}(1 - Pr(q_k))+$
$...+$
$\prod_{j=1}^{i} Pr(q_j) \times \prod_{k=i+1}^{n} Pr(q_k)=$
$\prod_{j=1}^{i} Pr(q_j) \times (1 - Pr(q_{i+1}) + Pr(q_{i+1})) \times \prod_{k=i+2}^{n}(1 - Pr(q_k))+$
$...+$
$\prod_{j=1}^{i} Pr(q_j) \times \prod_{k=i+1}^{n} Pr(q_k)=$
This is a recursive formula, and finally there only
$= \prod_{j=1}^{i} Pr(q_j)$.
which is $\prod_{q_j \in P_i} Pr(q_j) = Pr(A_{P_i})^{pos}$.

□

As a result, comparing the simulation relation between deterministic pd-processes can be achieved as follows: (i) first build the partition automata and compute their probability $Pr(A_{P_i})^{pos}$; (ii) finally, test the simulation relation of the partition automata. Because the closure automata are not generated, the expression complexity is based on the size of partition automata which is in EXPTIME.

Pseudo-deterministic pd-process

The definition of pseudo-deterministic pd-processes is as follows.

Definition 5.5 (Pseudo-deterministic pd-processes)**.** Let $A = \{S, s_0, D, Act, \Delta, G, F, AP, L\}$ be a pd-process, A is a pseudo-deterministic pd-process iff $\forall s_i \in S$ and $\forall (s_i, a_i, g_i, s_{i_1}) \in \Delta, \forall (s_i, a_j, g_j, s_{i_2}) \in \Delta$ such that $s_{i_1} \neq s_{i_2}$, $a_i = a_j$ and $g_i = g_j$.

Pseudo-deterministic pd-processes can be seen as a generalized case of deterministic pd-processes, because if the guards of transition relations with identical action are the same, these transition relations always occur in the same partition automaton, therefore it drops to the scope of determinism.

5.2 Strong non-deterministic pd-processes

In the previous section, the methodology of testing simulation relation between deterministic pd-processes or pseudo-deterministic pd-processes is studied but not all the pd-processes are pseudo-deterministic so in this section, the strong non-deterministic case will be described.

Definition 5.6 (Strong non-deterministic pd-processes)**.** Let $A = \{S, s_0, D, Act, \Delta, G, F, AP, L\}$ be a pd-process, A is a strong non-deterministic pd-process iff $\exists s_i \in S$ and $\exists (s_i, a_i, g_i, s_{i_1}) \in \Delta, \exists (s_i, a_j, g_j, s_{i_2}) \in \Delta$ such that $s_{i_1} \neq s_{i_2}$, $a_i = a_j$ and $g_i \neq g_j$. This kind of transition relations are called strong non-deterministic transition relations.

As the simulation relation test method described in Theorem 4.3, we have considered the intersections between all the partition automata so that there exist numeric volumes of abundant operations. The intersection between partition automata is valuable only when they are composed of strong non-deterministic transition relations. Because deterministic and pseudo-deterministic cases are captured by partition automata and their probabilities are recomputed through $Pr(A_{P_i})^{pos}$. Here we introduce the notion

of strong non-deterministic guards and deterministic guards as follows. We introduce a set named strong non-deterministic guards denoted as NDG. The guards which assign to strong non-deterministic transition relations are recorded in NDG. For other guards $g_j \notin NDG$, we call it a pseudo-deterministic guard. The set of pseudo-deterministic guards is denoted as DG. $NDG \cap DG = \emptyset$ and $NDG \cup DG = G$.To compute closure automata efficiently, two kinds of cases are considered as follows:

1. For a partition automaton A_{P_i}, $P_i \subset DG$, we do not consider it. Because the set DG does not contain any strong non-deterministic guards which means that there do not exist transition relations with same actions outgoing from one state. As a result, the intersections between A_{P_i} with other partition automata obviously follow the rule of Lemma 5.3.

2. During the intersections between two partition automata A_{P_i} and A_{P_j}, if $P_i \subset P_j$, we could ignore them because the result of $A_{P_i} \sqcap A_{P_j}$ is A_{P_i} and the probability of A_{P_i} has been captured in $Pr(A_{P_i})^{pos}$.

By ignoring these two cases, the algorithm to build closure automata is as follows:

Algorithm 6 Algorithm of computing closure automata under the context of strong non-determinism

Require:
One pd-process $A = \{S, s_0, D, Act, G, \Delta, F, AP, L\}$

Ensure:
1: For each state s_i, check if there exists strong non-deterministic transition relations.
2: If there exists, for all these strong non-deterministic transition relations and corresponding guards $\{g_{i_1}, g_{i_2}, ..., g_{i_j}\}$. Record all these guards into the set NDG.
3: For any guard partition $P_i \in P_G$, if $P_i \cap NDG = \emptyset$, delete P_i from P_G
4: Generate the set of closure automata $C_L(A_{P_G})$ which are the intersection with one or many partition automata in A_{P_G} when the involved guard partitions are $\{P_{i_1}, P_{i_2}, ..., P_{i_m}\}$ satisfying that any two guard partitions are not in this case: $P_{i_j} \subset P_{i_k}$. Then compute the probability $Pr(C_L(A_{P_G}))$.
5: **return** A set of closure automata.

Lemma 5.7. *Let $A = \{S, s_0, D, Act, G, \Delta, F, AP, L\}$ be a pd-process and DG is the set of pseudo-deterministic guards. The complexity of computing closure automata of pd-processes is in $O(2^{(2^{|G|}-2^{|DG|})})$.*

Proof. To prove this lemma, we compute the number of intersection operations to create closure automata. Because we don't need to compute the intersection between A_{P_i} and A_{P_j} if $P_i \subseteq |DG|$ and $P_j \subseteq |DG|$ according to Lemma 5.3, the number of candidate partition automata for closure automata is $2^{|G|} - 2^{|DG|}$. Then we compute the number

of intersections among these partition automata. Obviously, the complexity is the exponential of the size of involved partition automata. As a result, the complexity is in $O(2^{2^{|G|}-2^{|DG|}})$. □

From Lemma 5.7, we can find a fact that the complexity of computing closure automata relates to the size of deterministic guards, if $|DG|$ is 0, we get the upper bound of the complexity of computing closure automata in $O(2^{2^{|G|}})$; if $|DG| = |G|$, we don't need to compute closure automata.

Figure 5.1 illustrates the process of computing closure automata. Figure 5.1(a) shows an example of a pd-process with three guards G_1, G_2, G_3. Then we find 2 strong non-deterministic transition relaitons illustrated in Figure 5.1(b). Set NDG and DG are depicted in Figure 5.1(c). With the help of NDG and DG, we can generate the guard partitions denoted in Figure 5.1(d). Following Algorithm 6, the intersections between partition automata are made. Figure 5.1(e) illustrates some of the intersections and the result of these intersections are demonstrated in Figure 5.1(f).

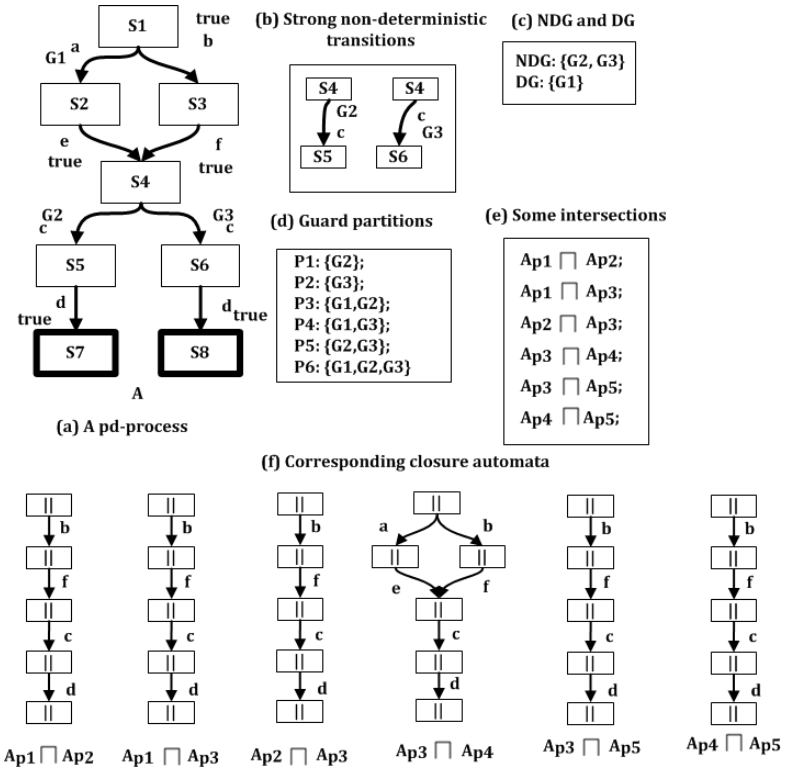

FIGURE 5.1: Example of computing closure automata

As a consequence of Lemma 5.7, we refine the complexity of testing simulation relation on pd-processes below. Let A and B be two pd-processes with G_A and G_B, the set of guards of A and B respectively and DG_A and DG_B, the set of pseudo-deterministic guards of A and B respectively. Then the simulation relation test between A and B can

be achieved in $O((2^{|G|} + 2^{(2^{|G|}-2^{|DG|})}) \times 2^{|G|})$. Say that if $|G_A| = |DG_A|$, the complexity is in EXPTIME.

5.3 Simulation test algorithm based on a compiled approach

In this section, we explain the fact that a lower-cost method to analyze a compiled approach that tests simulation relation between two pd-processes with data changes. For two given pd-processes$A$ and B, if guards and associated database instances are changed and we would like to check simulation relation again. The algorithm of previous section will take plenty of time. Because the structures of partition automata are not altered regardless probability, there is no need to compute them again. The guard partitions are not changed since the number of guards and transition relations keep the original arrange. In this case, we could reuse partition automata which are already generated and only need to recompute the probability of every partition automata or closure automata. So to test simulation relation of pd-processes in the context of guards' queries and database instances changed, Algorithm 7 depicts the first step about doing some pre-computation to generate partition automata and Algorithm 8 describes the method to test simulation relation of pd-processes when guards' queries or database instances are changed with the help of the result of pre-computation in Algorithm 7.

Algorithm 7 The compiled algorithm of simulation test

Require:

Two pd-processes $A = \{S, s_0, D, Act, G, \Delta, F, AP, L\}$ and $B = \{S', s'_0, D, Act', G', \Delta', F', AP', L'\}$ regardless the content of guards' queries or database instances.

Ensure:

1: Generate a set of guard partitions of A denoted as P_G.
2: According to P_G, build a set of partition automata denoted as A_{P_G}
3: Generate a set of closure automata denoted as $C_L(A_{P_G})$.
4: Follow the same manner to generate a set of guard partitions P'_G and partition automata $B_{P'_G}$.
5: For each A_{P_i}, record all the $B_{P'_i} \in B_{P'_G}$ such that $A_{P_i} \preceq B_{P'_i}$ into a mapping table T. T is a two column one-to-many mapping table where left column records $A_{P_i} \in A_{P_G}$ row by row and right column records a set of partition automata $\forall B_{P'_i} \in B_{P'_G} | A_{P_i} \preceq B_{P'_i}$ for a certain A_{P_i} in the left row. Do the same stuff for each $C_L(A_{P_i})$ and record their mappings in T. In T, if any element in left column maps with an empty set, the simulation test will return "False".
6: **return** Mapping table T or False.

To illustrate Algorithm 7 clearly, here is an example. Assuming two pd-processes A and B, partition automata of A are $A_{P_1}, A_{P_2}, A_{P_3}$ and a closure automaton $C_L(A_{P_1})$ whose

Algorithm 8 Algorithm of simulation test based on compiled approach

Require:

Two pd-process$A$ and B, mapping table T generated in Algorithm 7, a new database D and novel guards' queries.

Ensure:

1: If the pre-computation in Algorithm 7 doesn't return false,by using new database instance and novel guards' queries, the probability of each guard partition $P_i \in P_G$ and $P'_i \in P'_G$ should be recomputed as well as their corresponding partition automata. The probability of closure automata in $C_L(A_{P_G})$ will be updated by the sum of their associated partition automata.

2: In table T, for each row, we will check for each element in the left column, if there exists a partition automaton in the right column of the same row with equal or higher probability.

3: **return** True of Flase .

T	**A**	**B**
1	A_{P_1}	$B_{P'_1}, B_{P'_2}$
2	A_{P_2}	$B_{P'_2}, B_{P'_3}$
3	A_{P_3}	$B_{P'_3}$
4	$C_L(A_{P_1}) : \{A_{P_1} \sqcap A_{P_2}\}$	$B_{P'_2}$

TABLE 5.1: Mapping table of A and B

corresponding set of partition automata is $\{A_{P_1}, A_{P_2}\}$; partition automata of B are $B_{P'_1}, B_{P'_2}, B_{P'_3}$. We do some pre-computation and build the mapping table T as follows: In the first row of Table 5.1, left element is A_{P_1} and right element is $B_{P'_1}, B_{P'_2}$. It means A_{P_1} is simulated by $B_{P'_1}$ and $B_{P'_2}$ regardless probability. When guards and database instances are given, we refer to Algorithm 8 to compute the probability of each partition automaton and closure automaton of A, partition automaton of B. Then we go back to table T, we check the preorder relations w.r.t probabilities from left column to right column for each row. Each time when guards and database instances are changed, we can reuse the result of pre-computation and only refer to Algorithm 8 to recompute and check the preorder of the probabilities. Because Algorithm 8 recomputes the probability for each partition automata and closure automata. So it can be seen as a PTIME algorithm with respect to the size of partition automata.

5.4 Independent pd-processes

In pd-processes, guards $\{g_1, g_2, ..., g_n\}$ are boolean queries $\{q_1, q_2, ..., q_n\}$ on a given probabilistic database D returning "true" or "false". First, to evaluate the full relational algebra problem in probabilistic databases, we will reuse the methods proposed in [38]: each tuple of probabilistic database is annotated with a propositional formulas called

"events" over a set of boolean variables . Table 5.2 shows an example of several probabilistic tuples which add an additional attribute as events. The probability of the event owned by a tuple equals to the probability of this tuple. For the reason of simple management of query evaluation, we assume all the original tuples with atomic events are independent or disjoint and the other tuples derive from these original ones. The original atomic events are not only a signature of every tuple but also represent the lineage of every non-atomic tuple. The method to utilize the events annotated on probabilistic database for evaluating the SPJ queries is as follows:

If assuming this query uniquely impacts on two tuples t_1 with probability pr_1 and event X_1, t_2 with probability pr_2 and event X_2 from different tables, X_1 is a collection of atomic events which are independent or disjoint and considered as the original events, as well as X_2.

- **Selection from t_1 and t_2.** Selection never change the event and probability.
- **Projection from t_1 and t_2.** If projection impacts on both tuples, the result event is $X_1 + X_2$, the probability is the consequence of computing $X_1 + X_2$.
- **Join from t_1 and t_2.** If $t_1 \bowtie t_2$, result event is $X_1 \times X_2$, the probability is the consequence of computing $X_1 \times X_2$.

Announcing that t_1 and t_2 may be correlate tuples meaning that X_1 and X_2 have common atomic events. The computation of probability cannot be simply adding or product. For example, if $X_1 = y_1 + y_2$, $X_2 = y_1 + y_3$, then $X_1 + X_2 = y_1 + y_2 + y_3$. The probability of this result is $pr(y_1) + pr(y_2) + pr(y_3)$. With the same manner, if $X_1 = y_1 \times y_2$, $X_2 = y_1 \times y_3$, then $X_1 \times X_2 = y_1 \times y_2 \times y_3$. The probability of this result is $pr(y_1) \times pr(y_2) \times pr(y_3)$. We denote $E(q_i)$ as the formula of events generated by query q_i.

Definition 5.8 (Witness events)**.** For any boolean query q_i on a probabilistic database D, the witness events of q_i denoted as $we(q_i)$ are a set of atomic events such that $we(q_i) = \{e|\ e$ is an atomic event that occurs in $E(q_i)\ \}$.

According to Definition 5.8, if the set of witness events of q_i is empty, q_i returns "false". In the context of probabilistic database, the witness tuples are enumerated due to the computation of probability. In Table 5.2, if there is a boolean query :"Exists: select TID = '101' in T_1", $E(q_i) = x_1 + x_2$ the witness events $we(q_i) = \{x_1, x_2\}$.

Definition 5.9 (Independent guards)**.** Two guards g_1 and g_2 are independent guards w.r.t a fixed probabilistic database D iff $we(g_1) \cap we(g_2) = \emptyset$.

Original database

T_1	TID	Name	Pr	Event
	101	A	20%	x_1
	101	B	40%	x_2
	212	C	70%	x_3

T_2	SID	Name	Pr	Event
	133	A	50%	y_1
	134	D	60%	y_2
	276	C	40%	y_3

Query answer of $\Pi_{TID=101}T_1$

T_3	TID	Pr	Event
	101	60%	$x_1 + x_2$

Query answer of $\Pi_{Name}(T_1 \bowtie T_2)$

T_4	Name	Pr	Event
	A	10%	$x_1 \times y_1$
	C	28%	$x_3 \times y_3$

TABLE 5.2: Example of events annotated on probabilistic database

The definitions of independent guards is the principle criteria of determining if a pd-process can be decreased to a Markov process. The definition of independent pd-processes which capture the features of Markov processes is introduced as follows.

Definition 5.10 (Independent pd-processes)**.** A pd-process $A = (S, s_0, D, Act, G, \Delta, F)$ is independent pd-process if the following properties are satisfied.

1. the guards of all the transition relations started from a state s_i are independent guards.

2. for any two guards g_1 and g_2 whose corresponding transition relations are outgoing from different states, if g_1 and g_2 are in the path π, g_1 and g_2 are independent guards.

For the reason of simplifying notation, pd-processes indicate the normal pd-processes whereas independent pd-processes represent the ones with the restrictions of the guards' independence. As the descriptions in [44], a pd-process is a non-Markov process due to the correlation of the guards (maybe common witness tuples). But Definition 5.10 implies a fact that there dose not exist correlation among guards in the same independent guards closure or of the transition relations from same state in the context of independent pd-processes. As a result, a theorem can be deduced from Definition 5.10 as follows.

Theorem 5.11. *A pd-process is a Markov process iff it is an independent pd-process.*

Proof. $\Rightarrow$ The states, initial state and actions in pd-process own the parallel meaning with the ones of a Markov process. For any two transition relations δ_1, δ_2 stated from one state s_i in a pd-process A. Here we assume there are only two transition relations started from s_i. If A is a independent pd-process, the guards for δ_1, δ_2 are g_1, g_2, respectively

and obviously, $we(g_1) \cap we(g_2) = \emptyset$ for a given probabilistic database D.
If $E(D)$ represents the set of all atomic events in D, $Pr(g_1) = 1 - Pr(E(D) \setminus we(g_1))$, $Pr(g_2) = 1 - Pr(E(D) \setminus we(g_2))$ and assuming there exists an inactive transition relation δ_σ whose guard is g_σ with a set of witness events $we(g_\sigma) = E(D) \setminus (we(g_1) \cup ve(g_2))$, $Pr(g_\sigma) = 1 - Pr(E(D) \setminus \sigma) = 1 - Pr(we(g_1)) - Pr(we(g_2))$ such that $Pr(g_1) + Pr(g_2) + Pr(g_\sigma) = 1$, equally, $Pr(\delta_1) + Pr(\delta_2) + Pr(\delta_\sigma) = 1$ such that the set of transition relations started from s_i, $\Delta_{s_i} \subseteq s_i \times Act \times S$ can be represented as a probability space: $\Delta_{s_i} \subseteq s_i \times Probs(Act \times S) = Probs(s_i \times Act \times S)$ where $Probs(s_i \times Act \times S) = \{\delta_1, \delta_2, \delta_\sigma\}$ is a probability space (Ω, F, Pr) where Ω is $\{\delta_1, \delta_2, \delta_\sigma\}$, $Pr(\Omega) = 1$, F is discrete due to the exclusiveness of the elements in Ω.

Similarly as the step above, for a set of connected transition relations$s_1 \xrightarrow{a_1,g_1} s_2 \xrightarrow{a_2,g_2} s_3$, because g_1 and g_2 are in the same path so they have not common witness events, these two transition relations $s_1 \xrightarrow{a_1,g_1} s_2$ and $s_2 \xrightarrow{a_2,g_2} s_3$ are memoryless and discrete. This feature also coincides with the properties of Markov processes.

$\Leftarrow$ For any Markov process, if we consider all the probability distributions as guards, it obviously satisfies the two properties of an independent pd-process.

□

5.4.1 Simulation test of independent pd-processes

The simulation test algorithm in the context of independent pd-processes is trivial. Because an independent pd-process can be decreased to a Markov process, we first compute all the guards' probabilities, then we run the traditional probabilistic simulation test.

Complexity of simulation test of independent pd-processes

Let A and B be two independent pd-processes. The problem of checking whether $A \lesssim B$ is:
(i) in $\mathcal{O}(f(|D|))$ in data complexity, where $f(|D|)$ is the data-complexity of computing the probabilities of a boolean query on a probabilistic database D,
(ii) PTIME w.r.t. the expression complexity,
(iii) can be solved in PTIME w.r.t. $|A| + |B|$.

5.4.2 Model checking of independent pd-processes

Similar with the simulation test, we can transform independent pd-processes to Markov processes by computing the probability of each guard then run traditional P-LTL or P-CTL model checking algorithms as needed.

Complexity of model checking of independent pd-processes As Proposition 5.11 described, an independent pd-process is a finite Markov Chain. The model checking (LTL, CTL, P-LTL, P-CTL) algorithms of independent pd-processes can reuse the ones of finite Markov Chain (Markov processes).

(i) in $\mathcal{O}(f(|D|))$ in data complexity, where $f(|D|)$ is the data-complexity of computing the probabilities of a boolean query on a probabilistic database D,

(ii) the expression complexity is EXPTIME w.r.t P-LTL model checking, PTIME w.r.t P-CTL model checking.

iii P-LTL model checking can be solved in EXPTIME w.r.t $|\phi|$ (ϕ is a LTL formula).

iv P-CTL model checking can be solved in PTIME w.r.t $|\Phi| \times |A|$ (Φ is CTL formula).

Chapter 6

Related works

This chapter briefly describes related work of the main contents of this thesis and shows the differences between our contributions with these works. We mainly focus on the related domains as follows: data-centric/data-aware business processes, probabilistic processes, probabilistic database, verification method on probabilistic systems. The first part proposes the notion of data-centric/data-aware business processes and the second part focuses on the domain of probabilistic processes including modeling, simulation test, verification method in the context of probabilistic processes.

6.1 Data-centric/Data-aware business processes

Nowadays, experts do not only satisfy expressing business sequence of activities but also the informational perspectives which normally are treated as a part of context of single activities. The ideas of letting data play the most important role birthed data-centric/data-aware business processes. Normally, without specific explanations, data-centric business processes imply business artifact process. Intuitively, business artifacts are data objects whose manipulations define the underlying processes in a business model. Business artifact and information-centric processing of artifact life-cycles was first introduced in [45]. Then in [46], a business artifact processes in pharmaceutical research were designed and industrialized as a successful business attempt. Recently, in [47], a brand new framework based on business artifact process was proposed to integrate artifact-centric process model, process view model, a set of consistency rules, and the construction approach for building process views. The authors attempted to develop a bottom-up abstraction mechanism to construct process view by deriving from underlying process models and capturing major business requirements. Meanwhile, they defined a consistency rules to preserve the consistency between business views which were

constructed and their corresponding processes. It can be considered as a generalization of artifact-centric business processes.

Comparing with these formal works, we focus not only on the role played by data but also the flow perspective in the business processes. In business artifact processes, data is dissociated from the control flow perspectives and forms isolated views to illustrate the information exchanging and data transforming during the whole business processes. Our work directly integrates data by using guard querying probabilistic database in business processes. It clearly observes how data influence the transition relations among different states to predict future.

In spite of business artifact processes, data-aware conformance and compliance checking provide another way to consider the role of data played in the business processes. In [6], authors showed an abstraction approach to avoid state explosion by conducting compliance checking which is a part of process mining extracting information from event log for an abstract process model and abstract compliance rules. In [5], a more efficient method of data-aware conformance checking was extended from previous works by running a realistic BPMN processes ProM and evaluated using a variety of model-log combinations. The authors described an approach that aligns event log and model while taking all perspectives (data and resources) into account. An automatic composition of a web service called Colombo model was introduced in [48]. Colombo model is a framework in which web services are characterized in terms of the atomic processes which impact on the real world (modelled as a relation database). This framework is represented by following the notion of data-centric business process which is a control flow perspective integrating queries over relational databases as guards. In our work, we expand the notion of guards which query translational database in Colombo model to probabilistic guards querying probabilistic database. This change cause plenty of alternations with respect to the semantic of model, algorithms of simulation relation test and model checking. In [7], a formal specification and verification of data-centric service composition was represented to follow the paradigm of service oriented architecture. The authors use a running example of PayPal Express Checkout flow by applying their verification and specification approach. In their approach, the usefulness of a service contract at different level of sophistication was evaluated to facilitate a so called cost-benefit analysis. This analysis was decided on the time and effort invested by service providers in specifying their services. Meanwhile, the effectiveness of the specification in detecting programmers' errors while building applications by integrating services in an ad-hoc manner could be measure in their future works.

Up to our knowledge, this is the first work that discusses the integration between

probabilistic database and business processes. In our works, guards query probabilistic database instead of traditional databases. The result of these probabilistic guards provide a probabilistic predictions for every associated transition relation. As a consequence, our works can present a set of traditional processes with probabilistic distributions rather than one process. This feature will help analysers to predict the risks in the future by these probabilistic distributions.

6.2 Probabilistic process

Probabilistic process constitute a mathematical framework for the specification and analysis of probabilistic systems. The original work on probabilistic extensions of process algebras were published in [49], [50] by using labelled transition systems as an operational model in which probabilities are associated to transitions relations. In [51], the authors classified probabilistic models into three classes: reactive, generative and stratified. Different probability distributions are assigned to different actions in the reactive model. The probabilities assigned to the outgoing transitions of one state labelled sum up to 1. This can be considered as a coarse way to describe the notion of probabilistic space but the probability assigned in one transition relation does not affect the probability of successor states. Generative model assigned the probability to the states over all outgoing transition relations and stratified model considered branching structure in the context of generative model. In [52], the model of probabilistic automata was introduced by capturing the notion of probabilistic space. The semantic of probabilistic automata can be defined in terms of probabilistic space. With the help of definition of probabilistic automata, a transition relation from a state presents a probabilistic distribution over states rather than to a single state, coinciding with the definition of transition relations of ordinary automata. Thus, the non-deterministic choice among different transition relations can be considered as a probabilistic choice in the context of probabilistic automata is addressed in this paper. Meanwhile, the simulaiton test for probabilistic automata was given in this paper.

Besides the theories of probabilistic process, a flurry of applications of probabilistic process were implemented by researchers and engineers. In [53], the author discuss an application to verify PAR protocol with unreliable channels by reducing the process expression for the parallel composition of the protocol components to an expression of a fully probabilistic process on which the versification technique on probabilistic process was introduced. In [54], probabilistic process was used to predict the pavement condition rating and collect detail data in absence of sophisticated equipment or trained staff in a system called decision support system (DSS) to replace Pavement Maintenance Systems

(PMS). In DSS, the future condition of pavement can be predicted by calculating the collected data in terms of probability distribution of previous conditions and analysing these data in a business process. An application of analysing the accident from the collision of ships entering or leaving port by using probabilistic process was discussed in [55]. In this paper, the probabilistic distribution was assigned to semi-Markov chain to determine the probability of correct execution of critical manoeuvres during ship's entering and leaving the port as well as the probabilities of incorrect execution of critical manoeuvres by a ship, that leads to marine accidents.

Our work describes another kind of probabilistic process specification named as pd-processes which differ from these traditional models. We introduce the notion of probabilistic guards to the domain of probabilistic processes. In our works, guards are boolean queries over probabilistic databases to provide a probabilistic decision making for corresponding transition relations. Owing to probabilistic guards, the semantic of our pd-processes is changed from Markov perspective to non-Markov specification. A single pd-processes can represent a set of business processes which are described in terms of world-partition automata with probability distribution. Therefore, the algorithms of simulation relation and model checking are also changed to fulfil this semantic changes.

6.2.1 Simulation relation test

A simulation relation describes the containment relation of two finite state machines. Intuitively, a finite state machine simulates another one if it can match all of its moves. As a core stone of verification technique, simulation relation test provides a refinement and abstraction tool for verifying the properties of finite state machines. In [22], a polynomial algorithm of traditional simulation relation test was proposed and we use it to do the simulation test between two world-partition automata in our thesis. This algorithm is presented in Appendix. As to probabilistic processes, the simulation relation test was firstly studied in [52] as well as the model of probabilistic processes. The algorithm of simulation relation test in the context of probabilistic processes extended from the traditional simulation test algorithm in [22] but probability is a principle role to be considered. In [52], authors introduced two notions of simulation relations strong probabilistic simulation and branching probabilistic simulation with respect to probabilistic processes model. Strong probabilistic simulation ignores the internal probabilistic spaces which are the probabilistic spaces for every state but considers the global probabilistic space of a probabilistic process. Branching probabilistic simulation concerns every internal transition relations and states. Authors also proved that probabilistic simulation are compositional and preserve P-CTL formulas and P-CTL formulas without negation and existential quantification, respectively. Regarding our work, we reuse

the algorithm of traditional simulation relation test rather than probabilistic ones. That is because in the context of our work, a pd-process can represent a set of transitional finite state machine in terms of world-partition automata assigned with probabilities. In every world-partition automaton, the probability for each transition relation is the same value. So when we attempted to check simulation relation, we consider the value of probability (bigger or equal) first then check the simulation relations. Traditional simulation relation test is adequate to check the similarity among world-partition automata. But the volume of world partition automata is exponential with respect to the size of guards, the simulation relation test over pd-processes is in EXPTIME rather than PTIME.

6.2.2 Model checking

There are a range of different techniques for formal verification. Model checking is particularly well-fit for the automated verification of finite state systems. In the over thirty years since its invention, model checking has achieved multiple breakthroughs, extensively used in the hardware industry, and has become feasible for verifying many types of software as well. By [17], the definition of model checking can be defined as follows: "model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model". The finite-state model is usually described in terms of concurrent system (normally automata) and the properties are depicted by translating natural languages to temporal logical formulae. The pioneering works which focused on temporal logical modelling checking on finite-state system by introducing linear temporal logic (LTL) were done in [56], [57] and [58]. Meantime, computational tree logical model checking (CTL) was introduced in [59]. To fulfil the need of analysing uncertainty in various domain of scientific data management, probabilistic systems (probabilistic database, probabilistic process, etc.) were birthed. Following this trend, verification techniques for probabilistic models were occurred in eighties last century. The original work was proposed in [60] by introducing a graph-based algorithm to prove not 100% sure termination for finite state probabilistic processes. Then, the verification of qualitative ω-regular properties in terms of P-LTL formula for finite state system was first studied in [41, 61] by representing ω-regular properties in the form of deterministic non-bushi automata. In these papers, the qualitative LTL model checking problem was proved in PSPACE-complete. In [62], a double exponential lower bound was found for the problem of verifying a fact that a finite state automaton was capable for a given P-LTL formula. The problem of fairness in the context of P-LTL formula was studied in [41, 61] by expressing restrictions on the resolution of non-determinism in finite state

automata. A branching-time logic for probabilistic systems has been originally proposed in [63]. Later on, a probabilistic computational tree logical model checking algorithm was proposed in [64]. Authors gave algorithms for checking a concurrent system which is in terms of Markov chain satisfy a given qualitative properties. As a milestone work in the domain of probabilistic processes and verification technique in terms of computation tree logic, [52] presented not only the foundation of probabilistic processes but also action-based variants of P-CTL. The probabilistic model checker was flurry based on the various theories of probabilistic process model checking. The first prototype of P-CTL model checkers has been proposed in [65]. Then PRISM [66], ETMCC [67], MRMC [68] were birthed to fulfil the various needs of P-CTL model checking for multi models such as Markov chain,Markov decision processes, transition matrices, etc. LiQuor [69] is an alternative model checkers for Markov decision processes to verify quantitative properties in this context with SPIN [70] inside as the LTL model checker.

Our work extends from these formal works. Because we are in a probabilistic system, P-LTL and P-CTL are considered in our case. The syntax and satisfaction relations of P-LTL and P-CTL are reused in our work. But the semantic of our work is different from the traditional finite state machine. The algorithm of P-LTL and P-CTL model checking are refined to fulfil our needs. The algorithm of P-LTL model checking on pd-processes reuses traditional LTL model checking method for every world partition automaton. The level of complexity dose not increased still EXPTIME but the exponent of the complexity is the sum of the size of guards and P-LTL formulas rather than temporal logic formulas. Similarly, P-CTL model checking on pd-processes follows the method of traditional CTL model checking (sparse tree, searching for specification sets for every node of the sparse tree) but due to the exponential size of world partition automata, the complexity of P-CTL model checking over pd-processes is EXPTIME rather than PTIME.

Chapter 7

Prototype and Application

This section describes a prototype implementation on an application of pd-processes. Section 7.1 describes a prototype to model the pd-processes and to test the simulation relation between two pd-processes. The results of performance experiment on syntactic data is illustrated in Section 7.2. Finally, Section 7.3 attempts to use the methodology of pd-processes to model the behaviour of a realistic scenario.

7.1 The PRODUS Prototype

This prototype, called "PRODUS" [1], a **PRO**babilistic **D**ata-aware b**U**siness process **S**imulation framework, allows modeling the pd-processes and testing the simulation relation between two pd-processes. Fig 7.1 shows the screenshot of PRODUS. The programming language of PRODUS is $C\sharp$ and the probabilistic database is powered by PostgreSQL sever.

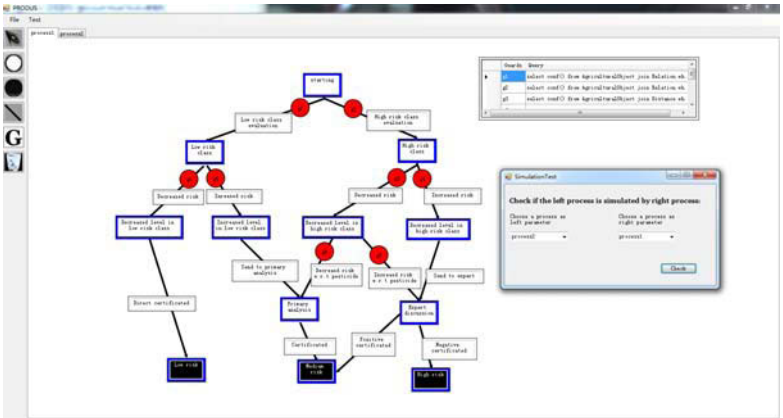

FIGURE 7.1: The PRODUS graphical user interface.

[1]The link of PRODUS: http://fc.isima.fr/~li/Application.html

Fig 7.2 depicts the global architecture of PRODUS which contains three main modules: the pd-process designer, the simulation handler, and the probabilistic database management system.

- Pd-process designer: there are three components in this module.
  - GUI: providing a graphical interface (PD-process model editor) to help users to generate pd-processes visually.
  - Pd-process model handler: (i) State handler: transferring the states descriptions in the GUI to an array-list. This array-list is used to store the input of the states; (ii) File handler: reading or writing files; (iii) Guard handler: querying probabilistic database and returning probabilities; (iv) Transition handler: Linking states and recording guards positions.
  - Partition generator: generating guard partitions, partition automata and closure automata.
- Simulation handler: testing simulation relation between two input pd-processes.
- Probabilistic database management system: executing boolean queries on PostgreSQL sever.

The process of modelling and testing simulation relation between two pd-processes in PRODUS requires (i) building the transition system of the pd-processes by connecting the states with transitions and specifying the contents of states, actions and guards in the *pd-process editor* module; (ii) in the *model handler* module, constructing the data structure of pd-process and resolving the guards in the *guard handler* module to query the probabilistic database management system by executing a boolean query; (iii) the P-DBMS is developed as an extension of the PostgreSQL sever which is in charge of storing all the data with probability and evaluating probabilistic boolean query used in pd-processes. In these probabilistic tables, probability and associated events are considered as additional attributes; (iv) computing the probability of each member in partitions of guards and based on it generating the partitions automata as the core function of *partition generator* module; (v)*simulation comparator* module comparing simulation relation by reusing the data produced by previous module; (vi) finally, the data of pd-process is stored in an XML format or txt file by the *file handler* module.

7.2 Experiment

The environment of this experiment is as follows :

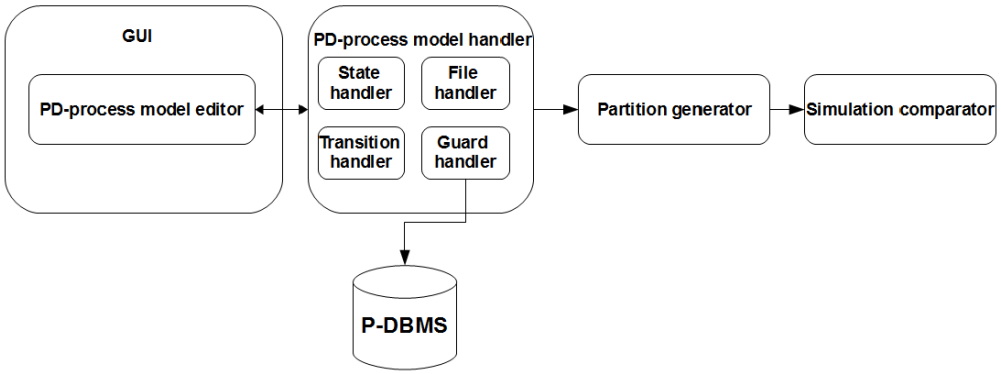

FIGURE 7.2: The PRODUS architecture.

- CPU: Intel(R) Core(TM) i5-2430M, 2.40GHZ, 4 cores.
- RAM: Maximum 2Gb.
- Hard disk: ST9500420AS-ATA.

7.2.1 Experiment of the optimized algorithm of simulation test

This subsection briefly describes the method of a performance experiment with the results through it. In the experiment, the running time and memory allocation of the prototype are concerned as the major elements. As a convincing result, it needs to execute the experiments numerous times. So an algorithm of generating syntactic pd-processes automatically is called up to fulfil this demand. Because of the homogeneous model of pd-process with other traditional automata, a modification on the automatic generation algorithm depicted in [71] is made to create typical automata rather than tree automata according to the feature of pd-process. Due to the query on the probabilistic databases by guards, automatic generation of the content of the guards is a new challenge. Following the results of the analysis in the previous sections, the contents of the probabilistic database do not give any impact to increase the complexity of the framework. So we could generate a naive probabilistic relation with only one attributes, as well as the probability distribution and homogeneous events for every tuple. Normally, users decide the number of tuple generated and the probability for each tuple is randomly assigned a number $N \in [0, 1]$. Table 7.1 illustrates an example of automatic generated probabilistic relation with 100 tuples.

With the help of the probabilistic relation generated as the method above, the content of a guard can simply copy the grammar as follows:

```
select * from [relation  name] where ID = '[random ID number]';
```

	ID	Pr	Events
t_1	1	0.43	x1
t_2	2	0.56	x2
...	...	...	...
t_{100}	100	0.12	x100

TABLE 7.1: Example of an automatic generated probabilistic database

Algorithm 9 depicts a method to generate a random pd-process. Because the names of states is meaningless for simulation relation test, all the states are generated as "S+the sequence number of states". Owing to comparing simulaiton between different pd-processes, the action names follow the same naming rule: "a+the sequence number of actions". As a result, same automatic generated pd-processes with the same or different size of alphabet pool can be used to test simulation relation.

Algorithm 9 Algorithm of automatic generation of pd-processes

Require:
Input state number N
Size of alphabet M
The scope of guards numbers

Ensure:
1: Pick the first generated state as initial state;
2: Randomly pick q numbers of state as final state with probability 0.5, $0 < q < N/2$;
3: For each state which is not a final state can randomly form a transition relation with other states (including itself) with a probability p_1;
4: Each transition relation picks up an action from the alphabet pool randomly and have a probability p_2 to own a guard;
5: The statement of the guard follows the grammar depicted as above;
6: **return** An automatic generated pd-process;

Fig.7.3 shows the result of the experiments. Fig.7.3(a) illustrates the memory allocation, noticing that when the guards number reaches 9, the memory allocation hits the ceiling. The time consuming of computing partition automata, closure automata and simulation test is shown in (b),(c) and (d), respectively. From these results, we can learn that: (i) the memory allocation strategy attempts to put more data in the main memory for avoiding I/O operations. Because the number of closure automata is 2-EXPTIME, taking 8 guards as an example, there are $2^{2^8} = 2^{256}$ closure automata theoretically, even after the optimization, the number decreases around 2^{70}. It still hit the ceiling of main memory; (ii) The average time consuming is proximately 2 hours when guard number is 10. The major problem is the high complexity of computing closure automata as previous tip noticed but there is not an efficient way to decrease the complexity of this step even the optimization is limited. The complexity of computing closure automata is still bounded between EXPTIME and 2-EXPTIME.

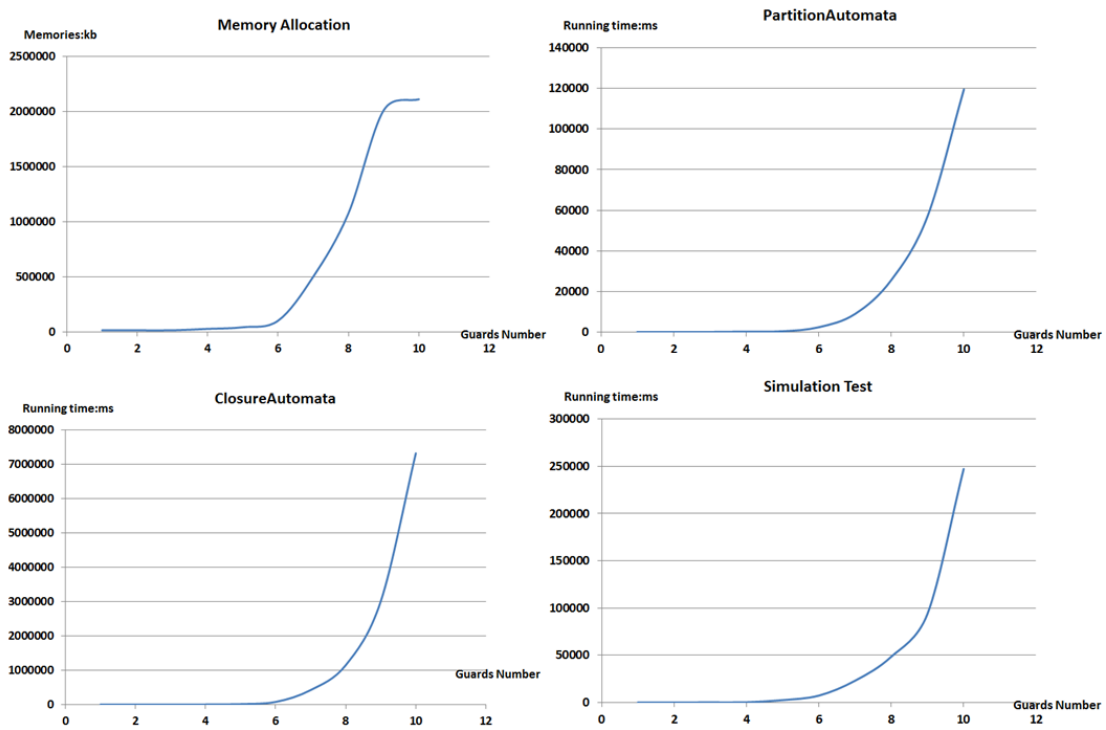

FIGURE 7.3: Experiment results.

7.2.2 Experiment of Algorithm 8

Because Algorithm 7 is the same complexity of the previous pd-processes simulation test algorithm, the experiments focus on the result of Algorithm 8. Fig 7.4 illustrates the results of experiments. Figure 7.4(a) shows the memory allocation w.r.t the number of partition automata. Because during the computation of Algorithm 8, only Table T is maintained in the main memory. The increase of memory allocation is linear and these is no need to record the details of each partition automaton. Figure 7.4(b) illustrates the running time. The most principle computation of Algorithm 8 is to query the probabilistic database. Since Table T is a hash table, the searching time can be considered as constant. So the time consume in this step is mainly cost during the manipulation of database.

Comparing with the result of previous experiment, based on a compiled approach, we only need to recompute the probability of each partition automata and closure automata. The expression complexity is EXPTIME with respect to the size of partition automata and closure automata and the data complexity is same as the one of querying probabilistic database.

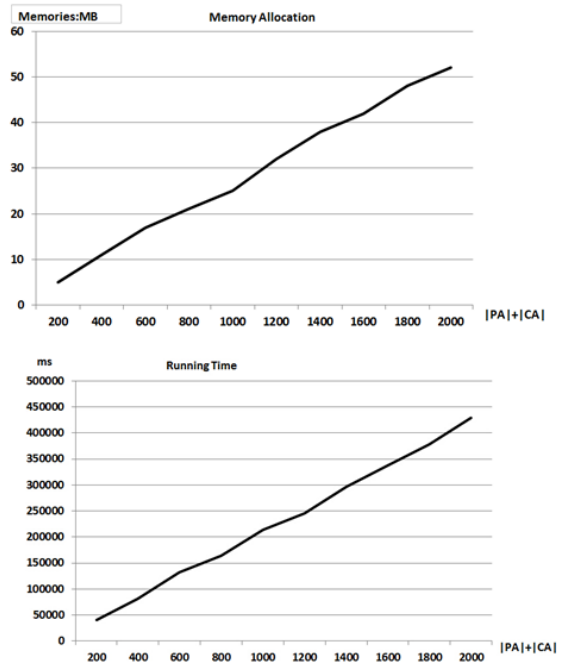

FIGURE 7.4: Experiment results of Algorithm 8.

7.3 Scenario

This subsection describes a realistic scenario application in the approaching of the agricultural field by collaboration with IRSTEA. IRSTEA is a research organization which, since more than 30 years, works on major issues of a responsible agriculture and territories sustainable planning, water management and related risks, drought, floods, inundations, the biodiversity and complex ecosystems study in their interrelation with human activities. We consider that combining the methods of probabilistic database, GIS (Geographical Information System) and business process modelling is a new trend to evaluate the agricultural activities. Pd-process is occasionally an ideal tool under this context. This scenario developed with IRSTEA shows how pd-processes could be used to evaluate the impact of agriculture activities more precisely. We attempts to evaluate the risk of agricultural activities among the hydrological objects (lakes, rivers, etc.) and agricultural plots. The spatial relation which provides information on the layout of spatial objects (distance, elevation, etc.) is the most principle information needed in the evaluation processes. Normally, this kind of information are uncertain and consequently the available data are often not precise so that probabilistic database is used to capture the uncertainty of the spatial objects' information in order to estimate the level of possible water and soil contamination (by agricultural inputs). Other than spatial relations, the type of farming, the varieties of crops in the farm, the pesticide utilization and other traits are also considered in this scenario to determine the risk level of agriculture activity on a certain farm during recent years.

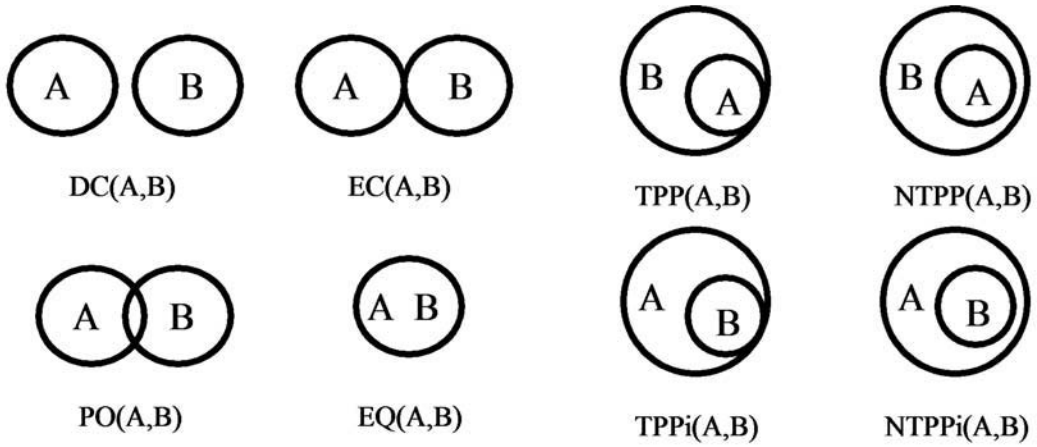

FIGURE 7.5: Spatial relations between simple 2-D regions defined in the Region Connection Calculus model

Spatial relations can provide information on the layout of spatial objects [72]. As shown in [73], there are different types of spatial relations. For example, topological relations are invariant under topological transformations. Metric information is provided by distance relations. Partial and total orders can be also modeled by spatial objects ("in the front of", "behind", etc.). Numerous spatial relations have been defined between different forms of spatial objects.

Probabilistic spatial relations can be defined between objects. This type of relations is associated to probability. In other words, we are able to assign probabilities on the different spatial relations.

Probabilities on topologic relations can be also deduce from a partial knowledge. To illustrate this type of methods, we introduce an example based on traditional binary spatial relations between simple 2-D regions (defined in the Region Connection Calculus model [74, 75] : Disconnected (DC), Externally Connected (EC), Equal (EQ), Partially Overlapping (PO), Tangential Proper Part (TPP), Tangential Proper Part inverse (TPPi), Non-Tangential Proper Part (NTTP), Non-Tangential Proper Part inverse (NTTPi). These relations are shown in Figure 2. All these relations are disjoint, i.e., there is only one spatial relation between two objects.

Suppose that we only know the three followings spatial relations between four simple 2-D regions:

- Relation 1: TPP(A, B).
- Relation 2: EC (B, C).
- Relation 3: TPP (C,D).

	R(A,B)	R(B,C)	R(C,D)	R(A,C)	R(A,D)	R(B,D)
Case1	TPP	EC	TPP	DC	DC	DC
Case2	TPP	EC	TPP	DC	DC	EC
Case3	TPP	EC	TPP	EC	DC	EC
Case4	TPP	EC	TPP	EC	DC	DC

TABLE 7.2: Deduction of new spatial relations

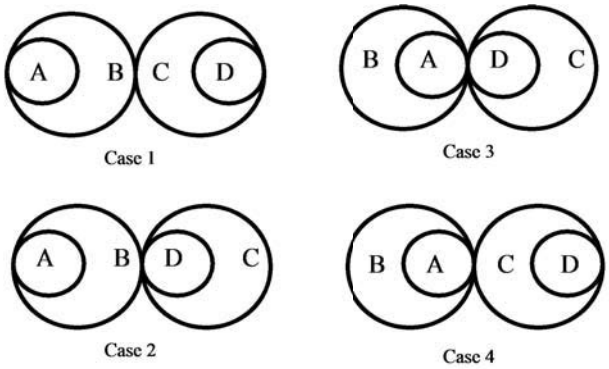

FIGURE 7.6: Four possible layouts

From the relations 1, 2 and 3, we can deduce new spatial relations between objects using the composition table presented in [74]. The results are in Table 7.2, Figure 7.5 shows an example of drawing for these spatial layouts. If we assume that all the rows of the table are equiproportional, we can determine different probabilities:

- $P(R(A,C) = DC) = 1/2$ and $P(R(A,C) = EC) = 1/2$;
- $P(R(A,D) = DC) = 1$;
- $P(R(B,D) = DC) = 1/2$ and $P(R(B,D) = EC) = 1/2$;
- The probability that two relations on the three unknown relations are EC is $1/4$, etc.

Table 7.3 shows a fragment of the probabilistic database of the spatial relation "Distance". The AGRICULTURAL_PLOT and HYDROLOGICAL_OBJECT tables record the necessary information for agricultural plots and hydrological objects. The DISTANCE_PLOT_HYDRO table presents the distance between one hydrological object and one agricultural plot with a probability. Our example database stores different information related to agricultural activities. Spatial representations of hydrological objects (lakes, rivers, etc.) and agricultural plots used for farming activities are in the database tables. These information are polygons. We suppose there is an intrinsic uncertainty on the boundary of the stored hydrological objects. For example, the hydrological limits evolve over the time. The measurement of the agricultural plots will also depend on the used acquisition techniques. So, we suppose that the same objects can have different

AGRICULTURAL_PLOT

ID_plot	Geo	Area	ID_farm	ID_soil_type	ID_watershed
$AP0031$	Poly{(11,32,79),(80,78,50)}	43.79	$Farm0734$	$Soil145$	$Warter0427$
$AP0702$	Poly{(68,70,67),(123,259,167)}	10.18	$Farm0197$	$Soil145$	$Warter0831$

HYDROLOGICAL_OBJECT

ID_hyd	Geo	Area	Name
$Hy0157$	Poly{(65,40,20),(90,108,67)}	103.79	L'Allier river
$Hy0470$	Poly{(209,408,89),(520,734,54)}	306.12	Citro forest

DISTANCE_PLOT_HYDRO

ID_plot	ID_hyd	Distance	Pr
$AP0031$	$Hy0157$	423.9	0.578
$AP0031$	$Hy0157$	320.7	0.235
$AP0702$	$Hy0157$	19.57	0.891

TABLE 7.3: An example of the agricultural spatial relation database schema.

representations. Consequently, the spatial relations between objects are uncertain. The different spatial object representations are not stored in our database, but the database contains the different spatial relations (with the associated probabilities) between these objects. In this application, we are interested in using the probability of spatial relations between agricultural plots and hydro-graphical areas in order to estimate the level of possible water and soil contamination (by agricultural inputs). The analysis of the probabilities of contamination levels can lead to different control actions. Because the spatial objects are uncertain, the distances between objects are also uncertain. Here we propose a generic database schema for probabilistic spatial relations that can be reused in many applications. A probabilistic spatial relation schema is composed of three types of tables:

1. Spatial objects table (SOT) stores information on spatial entities. SOTs can be probabilistic or not. SOT tuples contain: (i) a primary key; (ii) (optional) descriptive attributes;(iii) one (optional) spatial attribute (point, line, polygon, vague shape, fuzzy spatial objects, etc.) and (iv)(optional) foreign keys. The primary key can be composed of foreign keys or not.

2. Descriptive table (DT) stores information which are non-spatial by nature or which could have a spatial representation but that are not important for the application. DTs can be probabilistic or not. DT tuples contain: (i) a primary key; (ii) (optional) descriptive attributes;(iii)(optional) foreign keys. The primary key can be composed of foreign keys or not.

3. Spatial relations table (SRT) stores information on the probabilities of binary spatial relations. SRTs can be probabilistic or not. SRT tuples contain:(i) a primary key composed of the foreign keys coming from two SOT tuples; (ii) one (optional) spatial relation. This attribute is optional because the spatial relation

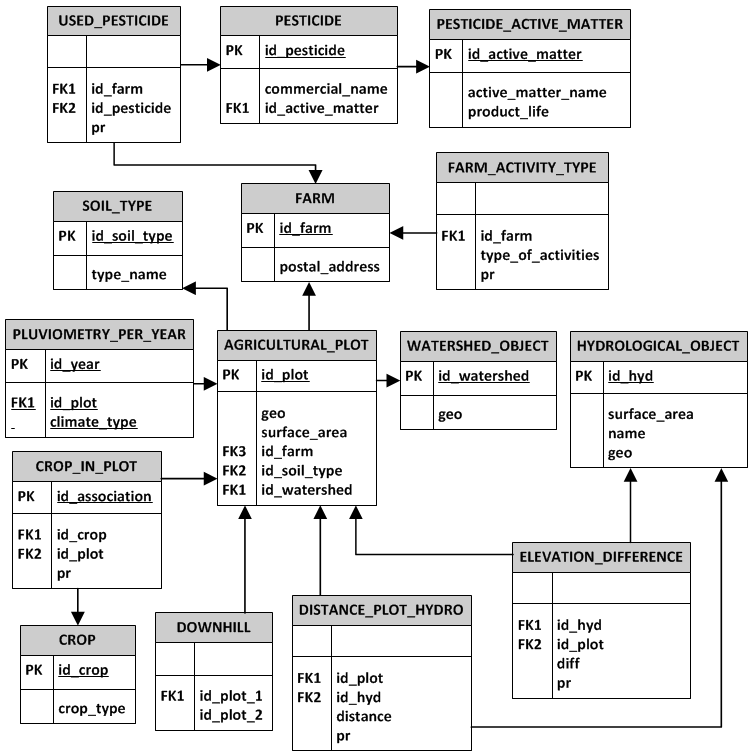

FIGURE 7.7: Database structure of scenario

can be implicit, as in following the SRT: LEFT_OF(obj_a:numeric,obj_b:numeric) – in this example no additional attribute are needed to specify the spatial relations ; obj_a is on the left of obj_b (and consequently obj_b is on the right of obj_a). The probability provides information on uncertainty of the spatial relation between two spatial objects.

In Figure 7.7, the explanation and clarification of database tables are as follows.

SOT

- AGRICULTURAL_PLOT
- HYDROLOGICAL_OBJECT
- WATERSHED_OBJECT

The information system considered in this example stores different plots used for farming activities and a set of hydrological objects such as lakes, rivers, etc. Different watersheds are considered.

DT

- FARM

- USED_PESTICIDE
- PESTICIDE
- PESTICIDE_ACTIVE_MATTER
- FARM_ACTIVITY_TYPE
- CROP
- CROP_IN_PLOT
- SOIL_TYPE
- PLUVIOMETRY_PER_YEAR

The plots are owned by farms. Each farm has one type of activities that can be intensive, semi-intensive or extensive. We consider here that we do not have access to information at the finest geographical level (i.e., for each farm), but aggregated information are available, such as the number of (semi-)intensive and extensive farms at regional level, for large, intermediate and small farms, etc. So, a probability can be estimated for each farm, depending on its properties (location, size, etc.). In the same way, the probabilities of crops cultivated over the years on plots can be estimated. Probabilities on pesticides used by farms can be also assessed. Geological information are available on the soil of plot. These information are used to classify the plots depending on its capacity to facilitate runoff or soil infiltration of water and pesticides. Pesticide run off is partially caused by rains. Global information on pluviometry is also provided.

SRT

- DISTANCE_PLOT_HYDRO
- ELEVATION_DIFFERENCE
- DOWNHILL

Elevation differences and distances (in meters) between plots and hydrological objects are stored in DISTANCE_PLOT_HYDRO and ELEVATION_DIFFERENCE (only for objects which are close). Data are stored in ELEVATION_DIFFERENCE only when plot are in downhill of hydrological objects. If one plot is in downhill of another plot, this spatial relation is stored in the DOWNHILL table.

Fig. 7.8 illustrates the processes of the agricultural risk evaluation modelled by a pd-process. One process evaluates the risk level of one agricultural plot with its nearby

hydrological objects by considering the spatial relations (distance, elevation, etc.), farming type, downhill and other traits. The risk level is divided into five different priorities and the probabilities calculated by different probabilistic database instances are provided by the guards to help decision-makers to decide to perform or not controls on a plot. Fig. 7.9 presents another two agricultural activities risk evaluation processes to compare with the previous. Fig. 7.9(A) shows the process to evaluate the grape planting farms and Fig. 7.9(B) demonstrates the apple cultivating farms evaluation process. To avoid creating extra spaces, the simulation test is used to determine whether the expressive power of one process is contained by another one. So we did simulation test of the processes in Fig. 7.9 with the process in Fig. 7.8. The result in Fig. 7.9(A) is simulated and Fig. 7.9(B) returns negative result. That is because the process in Fig. 7.8 considers general cases and grape planting are fortunately in this scope. Moreover, pesticides are widely used when cultivates apples. Experts should investigate every non-biometric samples manually to determine the pollution level.

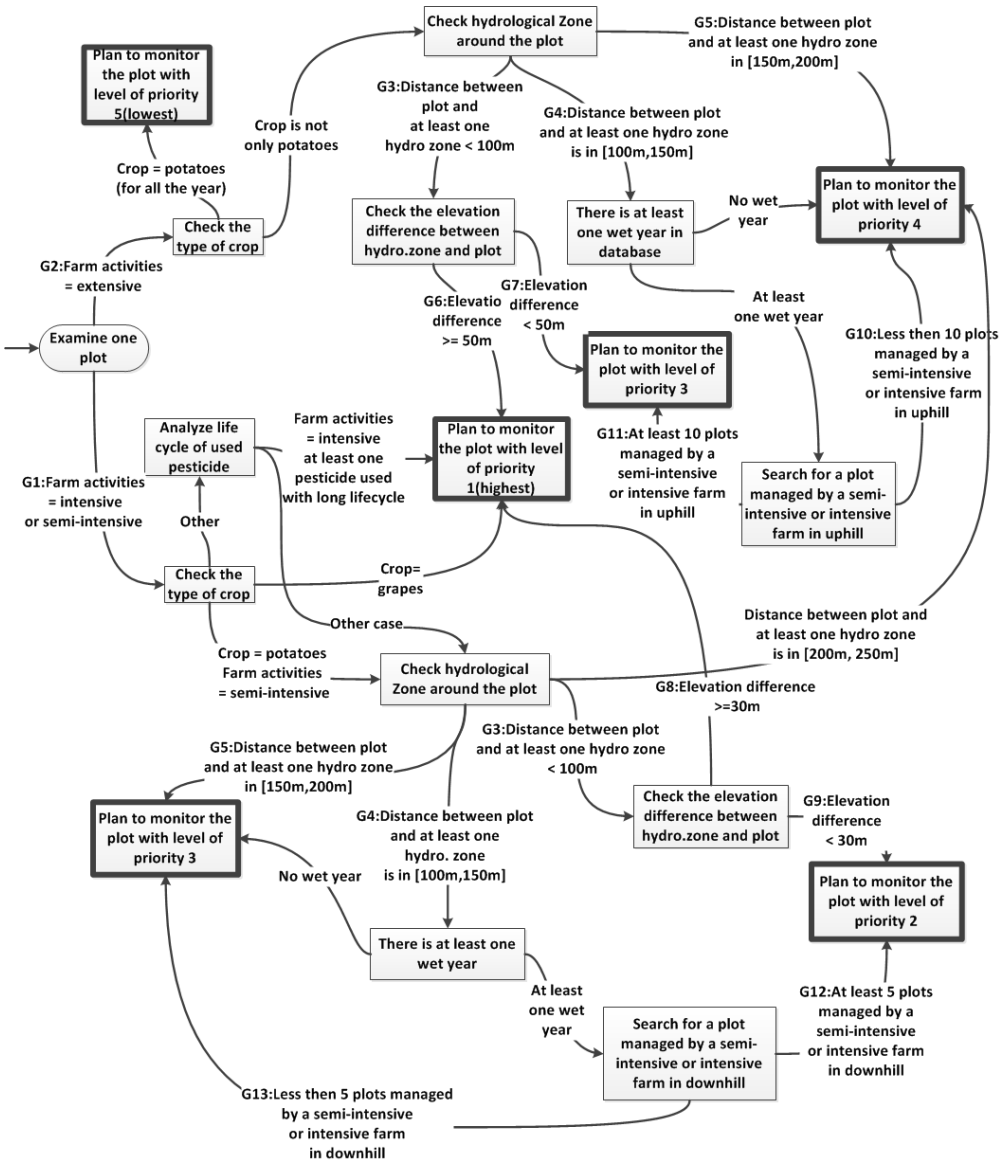

FIGURE 7.8: An agricultural activities risk evaluation process

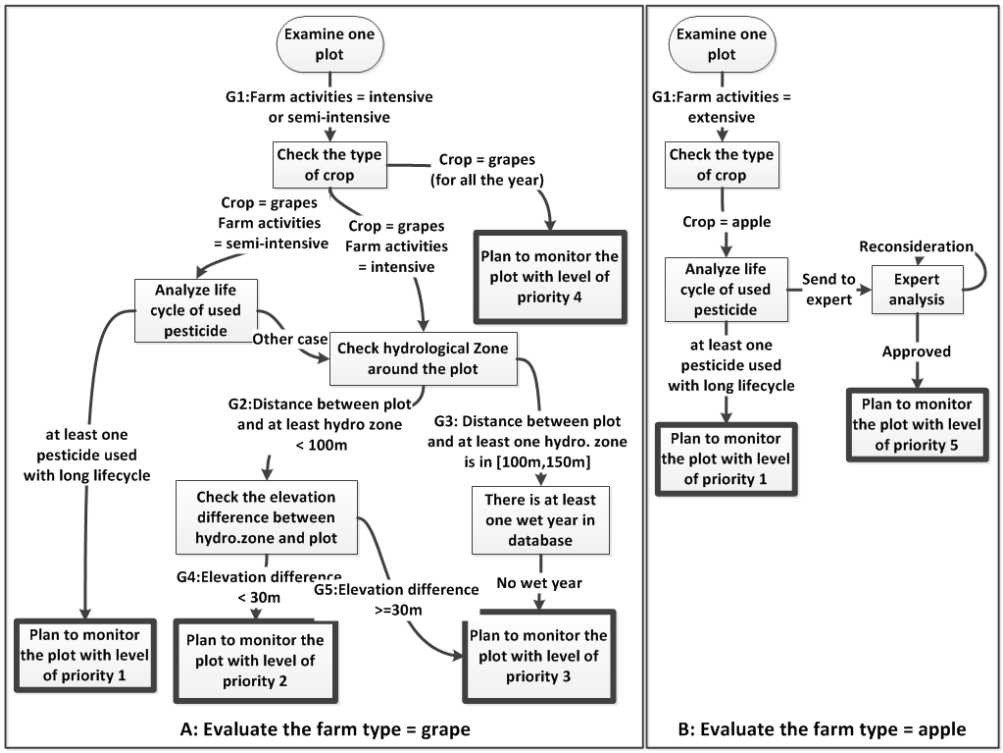

FIGURE 7.9: Another two agricultural activities risk evaluation processes

Chapter 8

Conclusion and Perspective

This thesis demonstrates a new model integrating probabilistic database and business processes named probabilistic data-aware business processes. The technique contributions focus on the formal definition of pd-processes the description of the semantic of pd-processes in terms of possible execution trees, the decomposition of pd-processes to partition automata, simulation relation test algorithm and model checking methods. Within the simulation test algorithm in the context of pd-processes, Theorem 4.3 describes a basic method and several optimized algorithms are proposed in the dimension of determinism, non-determinism of transition relations and compared approach of data alternation. We proved that the expression complexity of simulation relation testing over pd-processes is generally 2-EXPTIME and we can optimize it approaching EXPTIME (EXPTIME is the lower bound of testing simulation relation over pd-processes). Model checking methods focus on P-LTL and P-CTL formulae which meet pd-processes. The complexity of P-LTL checking dose not exceed the one of traditional probabilistic systems, still in EXPTIME. But the complexity of P-CTL model checking on pd-processes is in EXPTIME rather than PTIME in traditional cases. After verification methods, a way to identify pd-processes to Markov processes is discussed. We propose the notion of independent pd-processes which has a lower complexity of verification methods. Our work is implemented in the PRODUS prototype and experimented in the agricultural domain.

Perspective

The management of probabilistic data will be an increasingly important area over the next several years as businesses, governments, and scientific researchers contend with an ever-expanding amount of data. Although the space of applications will be diverse, there will be fundamental primitives common to many of these applications (just as there with standard, deterministic data). As a result, there will be a need for a general

purpose probabilistic data-aware business process management frameworks which can model, verify and analyse probabilistic data in a control-flow perspective. Our work will focus on enriching the functions of PRODUS, integrating more verification methods such as P-LTL and P-CTL model checking by reusing open sourced tools in it. Currently, our realistic scenario is only in the domain of agriculture but the research of probabilistic data has already reached various field such as sensor networks, health care, financial services or business intelligence. We will try to expand the usage of pd-processes in capable of more applications to forecast data, to tackle current data management problems and to help an analyst to interactively explore and understand a large collection of probabilistic data in a business process management system. Meanwhile, we will attempt to simplify the verification methods of pd-processes to decrease the complexity and dig on more properties. Probabilistic data-aware business processes are only one model of non-Markov processes by integrating databases into business processes. Investigating more general cases of data-aware business processes such as regarding different types of database such as incomplete database, uncertain database, etc.., will be a long term work.

Appendix A

Appendix

A.1 Traditional simulation relation test

The main idea of transitional simulation test is to check the containment of two finite state machines by considering actions assigned with the transition relation regardless the name of states. The efficient simulation test algorithm of finite state automata has been proposed in [22]. It is illustrated as follows:

For two finite state automata $A = (S, s_0, Act, \Delta, F, L, AP)$ and $A' = (S', s'_0, Act', \Delta', F', L'$ $, AP')$, to check if $A \lesssim A'$, we check for any δ, if there exist δ' and their simulator set $sim(\delta)$ and $sim(\delta')$ such that $\delta \in sim(\delta')$ and $\delta' \in \Delta'$. Simulator set $sim(\delta)$ stands for a set of transition relations $\{\delta'_i | \delta'_i \in \Delta'\}$ with the same action assigned with δ. The algorithm of simulation test is depicted in Algorithm 10. In this algorithm, the notation $pre(\delta)$ stands for the predecessor transition relations of δ, $presim(\delta)$ for all the candidates of $sim(\delta)$, $remove(\delta)$ for recording the transition relations removed from $presim(\delta)$, $delta''$ representing the transition relations in $remove(\delta)$. The complexity of this algorithm is in $O(mn)$ if $|m|$ is the size of A and n is the size of A'.

A.2 Model checking

Because in this thesis, PLTL model checking and PCTL model checking methods are not used. They are presented in the appendix.

Algorithm 10 Algorithm of simulation test

Require:
$A \lesssim A'$
Ensure:
for all $\delta \in \Delta$ do;
let $prevsim(\delta) = \Delta'$
if $post(\delta) == \emptyset$ **then**
  $sim(\delta) = \{\delta' \in \Delta' | Act(\delta) = Act(\delta')\}$.
else
  $sim(\delta) = \{\delta' \in \Delta' | Act(\delta) = Act(\delta') \&\& post(\delta') \neq \emptyset\}$.
end if
$remove(\delta) = pre(\Delta) n (sim(\delta))$
while there is $\delta \in \Delta$ such that $remove(\delta) \neq \emptyset$ **do**
  assert for all $\delta \in \Delta$, $remove(\delta) = pre(prevsim(\delta)) / pre(sim(\delta))$
  for all $\delta' \in pre(\delta)$ **do**
    for all $\delta'' \in remove(\delta)$ **do**
      if $\delta'' \in sim(\delta')$ **then**
        $sim(\delta') = sim(\delta') n \delta''$;
        for all $\delta'' \in pre(\delta'')$ **do**
          if $post(\delta'') \cap sim(\delta') == \emptyset$ then $remove(\delta') = remove(\delta') \cup \delta''$.
        end for
      end if
    end for
  end for
end while
let $prevsim(\delta) = sim(\delta)$.
$remove(\delta) = \emptyset$
return True or false.

A.2.1 P-LTL model checking

Following the definition of P-LTL, the method of applying P-LTL model checking is as follows:

1. A given probabilistic transition system $A = (S, s_0, Act, \Delta, AP, L)$ and LTL formula φ over AP checking $Pr(\varphi) >= c$ (c is a constant).

2. Do the regular LTL model checking and attempting to find a counterexample satisfying $Pr(\neg\varphi) >= 1 - c$.

3. Return true or false following a counterexample with probability.

A.2.2 P-CTL model checking

Similar method as CTL, we consider the 3 types of path formulae operators (next, until and bounded until)

- **Next**
  - Compute $Prob(s, \bigcirc\Phi)$ for all $s \in S$. $Prob(s, \bigcirc\Phi) = \sum_{s' \in Sat(\Phi)} P(s, s')$ where $P(s, s')$ presents the probability from s to s'.
  - Compute vector $Prob(\bigcirc\Phi)$ of probabilities for all the states.

  In fact, this algorithm equals to another method: compute the probability for every state, then find which states satisfy the given Φ not only the temporal formulae but also the probability restrictions.

- **Bounded Until and Unit** To compute the bounded until case denoted as $\pi \vDash \Phi_1 \cup^{\leqslant n} \Phi_2$, we have the method as follows:
  - Identify states with probability 1/0. As a result, we have $S^{yes} = Sat(\Phi_2)$, $S^{no} = S \setminus (Sat(\Phi_1) \bigcup Sat(\Phi_2)$, $S^{?} = S \setminus (S^{yes} \bigcup S^{no})$.
  - Then we compute the solution of recursive equations, $Prob(s, \Phi_1 \cup^{\leqslant k} \Phi_2) =$
    - 1, if $s \in S^{yes}$.
    - 0, if $s \in S^{no}$.
    - 0, if $s \in S^{?}, k = 0$.
    - $\sum_{s' \in S} P(s, s') \times Prob(s', \Phi_1 \cup^{\leqslant k} \Phi_2)$.
  - similar with "next" operator, we compute the result of $Prob(s, \Phi_1 \cup^{\leqslant k} \Phi_2)$ by matrix-vector multiplication. By contrast, there needs k times matrix-vector multiplication.
  - Until can be seen as a Bounded Until without bound.

To check if $M \vDash \Phi$, if M is a probabilistic process, the complexity is $O((|M|) \times n_{max} \times |\Phi|)$ where n_{max} is the maximal step bound in a sub-path formula $\Phi_1 \cup^{\leqslant n} \Phi_2$.

A.3 Algorithm of intersection of probabilistic boolean queries

Because in the current state of art, there is not a tool capable to handle the intersection of probabilistic boolean queries.But in our case, the intersection of probabilistic boolean queries is principle to build partition of guards or partition automata. So we need to implement our own algorithm of probabilistic boolean queries. The algorithm of intersection depicts in Algorithm 11:

Here is the explanation of the Algorithm 11.

Algorithm 11 Algorithm of intersection of probabilistic boolean queries

Require:
Two probabilistic boolean queries q_1 and q_2;
q_1 intersect with q_2.
Ensure:
if $q_1 = \emptyset$ or $q_2 = \emptyset$ **then**
  return Pr = 0
else
  Rewrite q_1 and q_2 as R_1 and R_2, respectively;
  $E_1 = \Pi_{Event} R_1$, $E_2 = \Pi_{Event} R_2$;
  $E_3 = E_1 \times E_2$;
  $E_4 = Merge(E_3)$
  $E_{trim} = Distinct(E_4)$
  $Pr(E_{trim}) = 1 - \prod_{\forall X_i}(1 - pr_{X_i})$, X_i is the events of E_{trim}.
  return E_{trim} and $Pr(E_{trim})$;
end if

- **Rewrite** Rewrite q_1 and q_2 returning tables of query answer rather than boolean values. This rewrite returns two relations R_1 with a collection of tuples$\{t_{11}, t_{12}, ...t_{1i}\}$, and R_2 with a collection of tuples$\{t_{21}, t_{22}, ...t_{2j}\}$, for each tuple t_{mk}, m = 1 or 2 , a probability pr_{mk} and an event X_{mk} associating with it, respectively.

- **Merge** E_3 Merge the two columns for each tuple of E_3 to form a new Relation E_4 which has only one attribute as follows: assuming X_1 is the value of first attribute and X_2 is the value of second attribute of E_3, Y_1 is the merge result such that $Y_1 = X_1 \times X_2$ and each atomic event in X_3 is distinct.

- **Distinct** E_4 Distinct the tuple of E_4 as follows: if Y_1 and Y_2 are the values of different tuples of E_4, considering Y_1 and Y_2 are two sets which contain a set of events as elements following the rules below: assuming y_1 and y_2 are two atomic events of Y_1,i if $y_1 \times y_2$, y_1 and y_2 are separated as two elements in set Y_1;ii if $y_1 + y_2$, they are considered as one element. Following this manner, the elements connecting with $\times$ are separated otherwise they are considered as an entirety by connecting with $+$. So if $Y_1 \subseteq Y_2$, delete Y_2. As a result, E_{trim} is build deriving from E_4;

Proof

This part attempts to prove Algorithm 11. Algorithm 11 describes a method to evaluate intersection of probabilistic boolean queries such that i rewriting the boolean queries to ordinary queries which return relations of query answer. This step ensures that all the tuples involved in this intersection query process are captured in the relations of query answer which rewrites the original boolean query, with the help of events which can be

considered as the lineage of the query. ii after rewriting, the probability of intersection of probabilistic boolean queries is computed according to lemma as follows.

Lemma A.1. *For given boolean queries q_1 and q_2, R_1 and R_2 are their rewriting relations of query answer, according to Algorithm 11, E_{trim} is built such that $Pr(q_1$ INTERSECT $q_2) = Pr(E_{trim})$.*

The proof of Lemma A.1 is explained as follows.

Proof.

- If PW_1 and PW_2 stand for the possible worlds of q_1 and q_2 respectively, the possible world of the intersection result is $PW_1 \cap PW_2$.
- Denote $T_1 = \{t_{11}, t_{12}, ..., t_{1n}\}$ as a set of tuples of the answer of q_1 and $T_2 = \{t_{21}, t_{22}, ..., t_{2m}\}$ as a set of tuples of the answer of q_2. For each tuple $t_{kl}, k = 1 or 2$, there is an event e_{kl} annotated, which contains a set of atomic events. This event e_{kl} presents a closure of possible worlds in which all the atomic events of e_{kl} are true, denoted as $CPW_{e_{kl}}$.
- As a result, $PW_1 = \bigcup_{i=n}^{1} CPW_{e_{1i}}$ and $PW_2 = \bigcup_{j=m}^{1} CPW_{e_{2j}}$. So $PW_1 \cap PW_2 = \bigcup_{i=n}^{1} CPW_{e_{1i}} \cap \bigcup_{j=m}^{1} CPW_{e_{2j}} = \bigcup_{i=n}^{1} (\bigcup_{j=m}^{1} (CPW_{e_{1i}} \cap CPW_{e_{2j}})$
- If we rewrite the query answer of q_1 and q_2 to two relations R_1 and R_2 rather than boolean values, such that $PW_{R_1} = PW_1$ and $PW_{R_2} = PW_2$. E_1 and E_2 present the set of events of R_1 and R_2. Obviously, E_1 is the projection on the attribute Event of R_1 as well as E_2. If we consider E_1 and E_2 are two sets, $E_1 = \{e_{11}, e_{12}, ..., e_{1n}\}$ and $E_2 = \{e_{21}, e_{22}, ..., e_{2m}\}$, $PW_{E_1} = \bigcup_{i=n}^{1} CPW_{e_{1i}} = PW_1$ and $PW_{E_2} = \bigcup_{j=m}^{1} CPW_{e_{2j}} = PW_2$.
- $E_3 = E_1 \times E_2 = \{(e_{11}, e_{21}), (e_{12}, e_{21}), ..., (e_{1i}, e_{2j}), ..., (e_{1n}, e_{2m})\}$
- $E_4 = Merge(E_3)$ such that $PW_{E_4} = \{CPW_{e_{11}} \cap CPW_{e_{21}}, ..., CPW_{e_{1i}} \cap CPW_{e_{2j}}, ..., CPW_{e_{1n}} \cap CPW_{e_{2m}}\}$.
- $E_{trim} = Distinct(E_4)$ such that $PW_{E_{trim}} = (CPW_{e_{11}} \cap CPW_{e_{21}}) \cup \cdots \cup (CPW_{e_{1i}} \cap CPW_{e_{2j}}) \cup \cdots \cup (CPW_{e_{1n}} \cap CPW_{e_{2m}})$
- Finally, $E_{trim} = PW_1 \cap PW_2$, Algorithm 11 is confirmed.

□

A.4 Spatial relation

In this section, two more methods to determine the probability of spatial relation is provided.

A.4.1 Method 1

Suppose five data sources. Data sources ♯1, ♯2, ♯3 store different spatial representations of the same objects A. Data sources ♯4, ♯5 store different spatial representations of the same object B. These different representations are caused by the uses of different measurement techniques. Fig. provides an example of the instances A and B of the five data sources; di is the different possible minimal distances between A and B. We consider that only one data source stores the correct representation for the object A (♯1, ♯2 or ♯3) - but we do not know which one is correct. In the same manner, only one data source stores the correct representation for the object B (♯4 or ♯5) - but the correct source is unknown. So, if we make the hypothesis that all the rows of Table A.1 are equiproportional events: $p(d1) = 1/6$, $p(d2) = 1/2$ and $p(d3) = 2/3$. In other word, if we do not know which data source stores the correct representation (for A and B), the probabilities that d1 is the correct distance is 1/6, etc. Note that the distance that minimizes the risk (i.e., the wrong choice) is $d2$. An order can be also provided between these distances ($d2 < d1 < d3$) and probabilities of other events can be calculated: for example, $p(<= d1) = 5/6$, i.e., the probability that the minimal distance is less than or equal to $d1$ is 5/6.

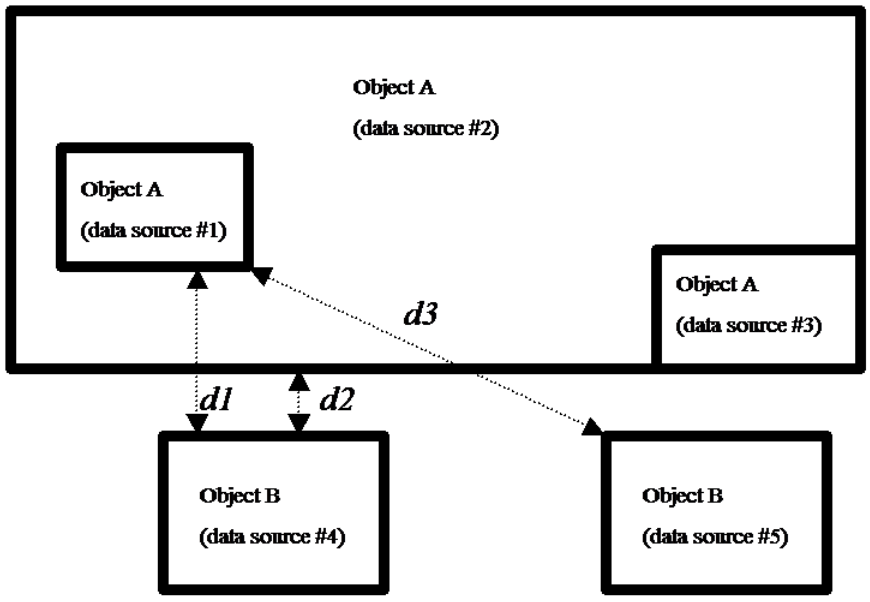

FIGURE A.1: Spatial representation of two objects represented in five data sources.

Source 1	Source 2	Source 3	Source 4	Source 5	Correct distance
correct	incorrect	incorrect	correct	incorrect	d1
incorrect	correct	incorrect	correct	incorrect	d2
incorrect	incorrect	correct	correct	incorrect	d2
correct	incorrect	incorrect	incorrect	correct	d3
incorrect	correct	incorrect	incorrect	correct	d2
incorrect	incorrect	correct	incorrect	correct	d2

TABLE A.1: Different hypotheses on the data source reliability

A.4.2 Method 2

Probabilistic spatial relations can be also calculated in the context of moving objects. Suppose that two commands are sent to a robot A controlled remotely:

1. A has to turn in an angle of 90 degrees.
2. Then, A have to walk 1 meter.

Due to the used robot material, the actions of the robot have a certain level of errors: $+/-[0..20]$ degrees when A turns and $+/-[0..20]\%$ for the covered distance. The possible positions for the robots are represented in Figure 4. The robot A is represented by a point drawn in bold. Pos1 is its initial position and Pos2 is its final position if the robot material has no error. If we consider the errors, the boundary of possible final positions (denoted by D) is represented by a polygon drawn in bold. This boundary is calculated using the maximal angle error and the maximal covered distance error. Consequently, the topological relations between A and the rectangular object O are uncertain. Probabilities can be calculated for these spatial relations by comparing the surface area of $D \cap O$ with the surface area of D.

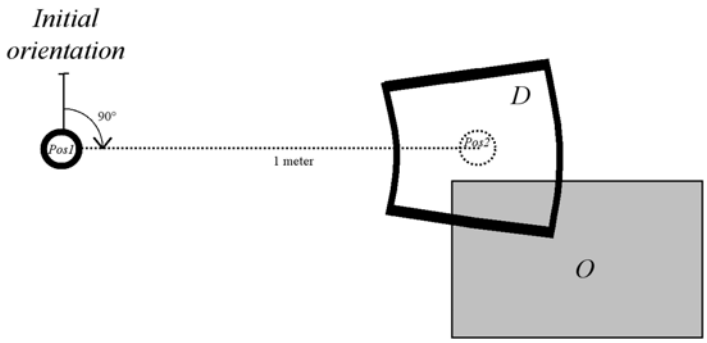

FIGURE A.2: Possible positions for the robot A.

Bibliography

[1] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012.

[2] Wil M. P. van der Aalst. Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013, 2013.

[3] Niels Lohmann. Compliance by design for artifact-centric business processes. *Inf. Syst.*, 2012.

[4] Elio Damaggio, Alin Deutsch, and Victor Vianu. Artifact systems with data dependencies and arithmetic. *ACM TODS*, 37(3):22:1–22:36, 2012.

[5] M. de Leoni, W. van der Aalst, and B.F. van Dongen. Data- and resource-aware conformance checking of business processes. In *BIS*, pages 48–59, 2012.

[6] D. Knuplesch, L.T Ly, S. Rinderle-Ma, H. Pfeifer, and P. Dadam. On enabling data-aware compliance checking of business process models. In *ER'2010*, pages 332–346.

[7] Iman Saleh, Gregory Kulczycki, and M. Brian Blake. Demystifying data-centric web services. *IEEE Internet Computing*, 13(5):86–90, September 2009.

[8] Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

[9] C. Rodriguez, F. Daniel, F. Casati, and C. Cappiello. Toward uncertain business intelligence: The case of key indicators. *IC*, 14(4):32 –40, 2010.

[10] Andreas Wombacher. A-posteriori detection of sensor infrastructure errors in correlated sensor data and business workflows. In *BPM 2011*, volume 6896, 2011.

[11] S. Tranquillini, P. Spieß, F. Daniel, S. Karnouskos, F. Casati, N. Oertel, L. Mottola, F.J. Oppermann, G.P. Picco, K. Römer, and T. Voigt. Process-based design and integration of wireless sensor network applications. In *BPM*, pages 134–149, 2012.

[12] C.A. Middelburg and M.A. Reniers. Introduction to process theory. Technical report, Technische Universiteit Eindhoven, 2004.

[13] Kais Klai, Samir Tata, and Jörg Desel. Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes. In *BPM*, pages 294–309, 2009.

[14] P.Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. *Proc. VLDB Endow.*, 1(1):809–820, August 2008.

[15] Todd J. Green. Models for incomplete and probabilistic information. In Charu Aggarwal, editor, *Managing and Mining Uncertain Data*. Springer, 2009.

[16] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *FMSD*, 6(1):11–44, 1995.

[17] Edmund. M. Clarke and Bernd-Holger Schlingloff. Model checking. In *Handbook of Automated Reasoning*, pages 1635–1790. Elsevier and MIT Press, 2001.

[18] Frank Puhlmann and Mathias Weske. Investigations on soundness regarding lazy activities. In *BPM*, BPM'06, pages 145–160, 2006. ISBN 3-540-38901-6, 978-3-540-38901-9.

[19] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *DKE*, 58(3):327–357, 2006.

[20] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, and Fabio Patrizi. Automatic service composition via simulation. *IJFCS*, 19(2):429–451, 2008.

[21] A. Sokolova and E. P. De Vink. Probabilistic automata: System types, parallel composition and comparison. In *Validation of Stochastic Systems: A Guide to Current Research*, pages 1–43, 2004.

[22] T.A. Henzinger, S. Qadeer, S.K. Rajamani, and S. Tasiran. An assume-guarantee rule for checking simulation. *ACM Trans. Prog. Lang. Syst.*, 24(1):51–64, Jan' 02.

[23] Jennifer Widom. Trio: a system for data, uncertainty, and lineage. *Charu Aggarwal, editor, Managing and Mining Uncertain Data, chapter 5.*, 2008.

[24] T. Imielinski and Jr W. Lipski. Incomplete information in relational databases. *J. ACM*, 31:761–791, 1984.

[25] P. Kanellakis S. Abiteboul and G. Grahne. On the representation and querying of sets of possible worlds. *Proc. SIGMOD*, pages 34–48, 1987.

[26] Simon Razniewski and Werner Nutt. Completeness of queries over incomplete databases. *Proceedings of the VLDB Endowment,*, 4:749–760, 2011.

[27] S.Abiteboul, L.Segoufin, and V.Vianu. Representing and querying xml with incomplete information. *ACM TODS*, 31:208–254, 2006.

[28] Erol Gelenbe and Georges Hebrail. A probability model of uncertainty in data bases. *Proc. 2nd IEEE Int. Conf. on Data Eng*, pages 328–333, 1986.

[29] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowl. and Data Eng*, 4:487–502, 1992.

[30] Norbert Fuhr and Thomas Rolleke. A probabilistic relational algebra for the integration of information retrieval and database syst. *ACM Trans. Inf. Syst*, 15:32–66, 1997.

[31] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM Trans. Database Syst*, 22: 419–469, 1997.

[32] Xin Luna Dong, Alon Halevy, and Cong Yu. Data integration with uncertainty. *Very Large Data Bases J*, pages 469–500, 2009.

[33] Andrew Nierman and H.V. Jagadish. Protdb:probabilistic data in xml. *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 646–657, 2002.

[34] Landon Detwiler, Wolfgang Gatterbauer, Brent Louie, Dan Suciu, and Peter Tarczy-Hornoch. Integrating and ranking uncertain scientific data. *Proc. 25th IEEE Int. Conf. on Data Eng*, pages 1235–1238, 2009.

[35] Lyublena Antova, Christoph Koch, and DanOlteanu. Maybms:managing incomplete information with probabilistic world-set decompositions. *Proc. 23rd IEEE Int. Conf. on Data Eng*, pages 1479–1480, 2007.

[36] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Prdb: managing and exploiting rich correlations in probabilistic databases. *Very Large Data Bases J.*, 18:1065–1090, 2009.

[37] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. *Proc. ACMSIGMOD Int. Conf. on Management of Data*, pages 551–562, 2003.

[38] Robert Fink, Dan Olteanu, and Swaroop Rath. Providing support for full relational algebra in probabilistic databases. *Proc. 27th IEEE Int. Conf. on Data Eng*, 2011.

[39] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. 1995.

[40] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR*, pages 481–496, 1994.

[41] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. *In 26th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 327–338, 1985.

[42] R.J. van Glabbeek. The Linear Time – Branching Time Spectrum (extended abstract). In *CONCUR'90*, volume 458 of *LNCS*, pages 278–297. Springer, 1990.

[43] M.Grabowski and S.Lasota. Complexity of simulation preorder for a product of finite automata. Technical report, University of Warsaw, 2010.

[44] H. LI, F. Pinet, and F. Toumani. Probabilistic simulation for probabilistic data-aware business processes. In *Language and Automata Theory and Applications*, volume 8370, pages 503–515. Springer International Publishing, 2014.

[45] A. Nigam and N.S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42:428–445, 2003.

[46] K Bhattacharya. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM SYSTEMS JOURNAL*, pages 145–162, 2005.

[47] Yongchareon. S and Liu. C. Process view framework for artifact-centric business processes. *OTM 2010, Part I. LNCS*, 6426:26–43, 2010.

[48] Daniela Berardi et al. Automatic composition of transition-based semantic web services with messaging. *31st VLDB Conference*, pages 613–624, 2005.

[49] C.C. Jou. Aspects of probabilistic process algebra. *Ph.D.Thesis, State University of New York at Stony Brook*, 1990.

[50] C.C. Jou and S. A.Smolka. Equivalences, congruences and complete axiomatizations for probabilistic processes. *Proc. CONCUR 90, LNCS 458, J.C.M. Baeten, J. W. Klop, eds. Springer Verlag*, pages 367–383, 1990.

[51] R. J. van Glabbeek, S. A. Smolka, B. Steffen, and C. M. N. Tofts. Reactive, generative and stratisfied models of probabilistic processes. *Proc. of 5th Annual IEEE Symp. on Logic in Computer Science*, pages 130–141, 1990.

[52] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nord. J. Comput.*, 2(2):250–273, 1995.

[53] Suzana Andova. Probabilistic process algebra. *Ph.D.Thesis, CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN*, 2002.

[54] Katkar Surendrakumar, Nagrale Prashant, and Patil Mayuresh. Application of markovian probabilistic process to develop a decision support system for pavement

maintenance management. *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*, 2:295–303, 2013.

[55] Jerzy Girtler. Application of theory of semi-markov processes to determining distribution of probabilistic process of marine accidents resulting from collision of ships. *Polish Maritime Research*, 21:9–13, 2014.

[56] Allen Emerson, E. Clarke, and Edmund M. Characterizing correctness properties of parallel programs using fixpoints. *Automata, Languages and Programming*, 1980.

[57] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. *Logic of Programs: Workshop, Yorktown Heights, NY*, 131, 1981.

[58] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, pages 244–263, 1986.

[59] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33:151–178, 1986.

[60] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5:356–380, 1983.

[61] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *In 1st Annual Symposium on Logic in Computer Science (LICS)*, pages 332–344, 1986.

[62] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42:857–907, 1995.

[63] S. Hart and M. Sharir. Probabilistic propositional temporal logics. *Information and Control*, 70:97–155, 1986.

[64] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6:512–535, 1994.

[65] L.-A. Fredlund. The timing and probability workbench: a tool for analysing timed processes. *Technical Report 49, Uppsala University*, 1994.

[66] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with prism: A hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6:128–142, 2004.

[67] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model-checking markov chains. *International Journal on Software Tools for Technology Transfer*, 4:153–172, 2003.

[68] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A markov reward model checker. *In 2nd International Conference on Quantitative Evaluation of Systems (QEST)*, pages 243–244, 2005.

[69] F. Ciesinski and C. Baier. Liquor: a tool for qualititative and quantitative linear time analysis of reactive systems. *3rd Conference on Quantitative Evaluation of Systems (QEST)*, pages 131–132, 2006.

[70] Gerard J. Holzmann. Design and validation of computer protocols. *PRENTICE-HALL*, page 1991.

[71] C. Nicaud, P. Heam, and S.Schmitz. Random generation of deterministic tree (walking) automata. In CIAA'09, editor, *Lecture Notes in Computer Science*, volume 5642, pages 115–124. Springer, 2009.

[72] J Freeman. The modelling of spatial relations. *Computer Graphics and Image Processing*, 4:156–171, 1975.

[73] Egenhofer. M and Herring. J. Categorizing binary topological relations between regions, lines, and points in geographic databases. *Department of Surveying Engineering, University of Maine, Orono, ME.*

[74] Cohn. A, Bennett. B, Gooday. J, and Gotts. N. Representing and reasoning with qualitative spatial relations about regions. *O. Stock (Ed.), Spatial and Temporal Reasoning*, pages 97–134, 1997.

[75] Randell. D. A, Cui. Z, and Cohn. A. G. A spatial logic based on regions and connection. *Proc. 3rd Int. Conf. on Knowledge Representation and Reasoning, San Mateo*, 1992.